LAB ASSIGNMNET 13.1

Subject : AI Assistant Coding

Hall Ticket No : 2303A52465

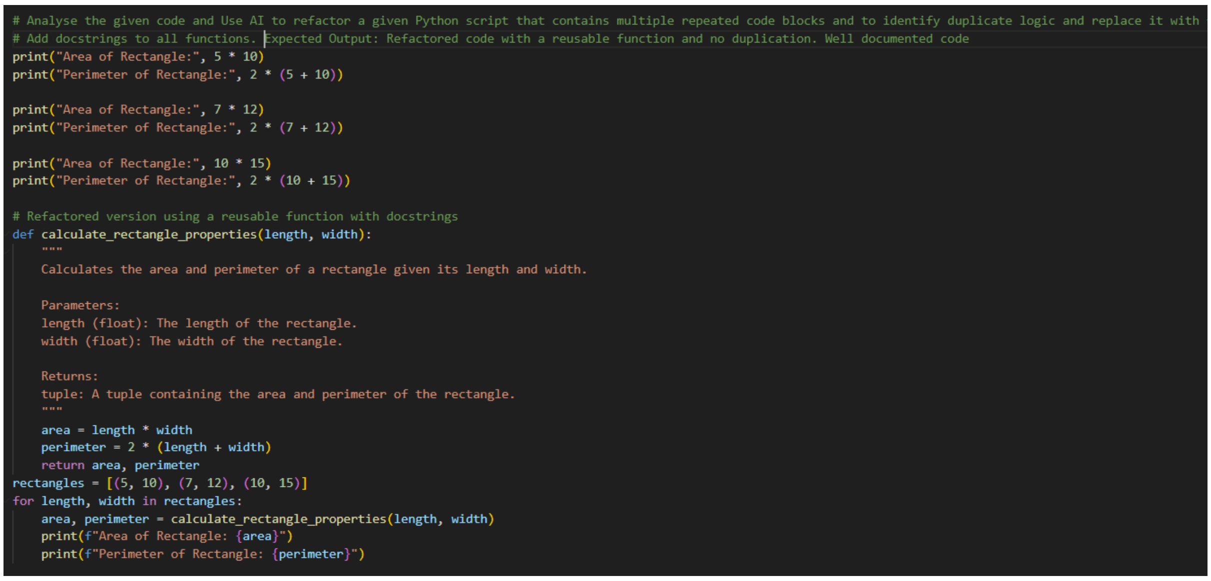
Name : M.Avinash

Batch : 50

Date : 27/2/26

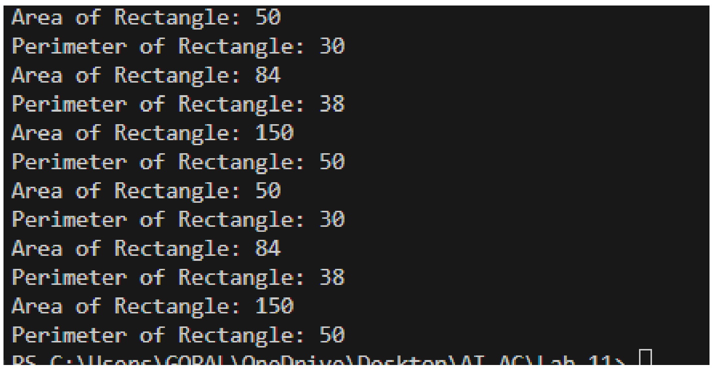
Task-1 :

Prompt : Analyse the given code and Use AI to refactor a given Python script that contains multiple repeated code blocks and to identify duplicate logic and replace it with functions or classes. Add docstrings to all functions. Expected Output: Refactored code with a reusable function and no duplication. Well documented code



```python
# Analyse the given code and Use AI to refactor a given Python script that contains multiple repeated code blocks and to identify duplicate logic and replace it with
# Add docstrings to all functions. Expected Output: Refactored code with a reusable function and no duplication. Well documented code
print("Area of Rectangle:", 5 * 10)
print("Perimeter of Rectangle:", 2 * (5 + 10))

print("Area of Rectangle:", 7 * 12)
print("Perimeter of Rectangle:", 2 * (7 + 12))

print("Area of Rectangle:", 10 * 15)
print("Perimeter of Rectangle:", 2 * (10 + 15))

# Refactored version using a reusable function with docstrings
def calculate_rectangle_properties(length, width):
    """
    Calculates the area and perimeter of a rectangle given its length and width.

    Parameters:
    length (float): The length of the rectangle.
    width (float): The width of the rectangle.

    Returns:
    tuple: A tuple containing the area and perimeter of the rectangle.
    """
    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter
rectangles = [(5, 10), (7, 12), (10, 15)]
for length, width in rectangles:
    area, perimeter = calculate_rectangle_properties(length, width)
    print(f"Area of Rectangle: {area}")
    print(f"Perimeter of Rectangle: {perimeter}")
```

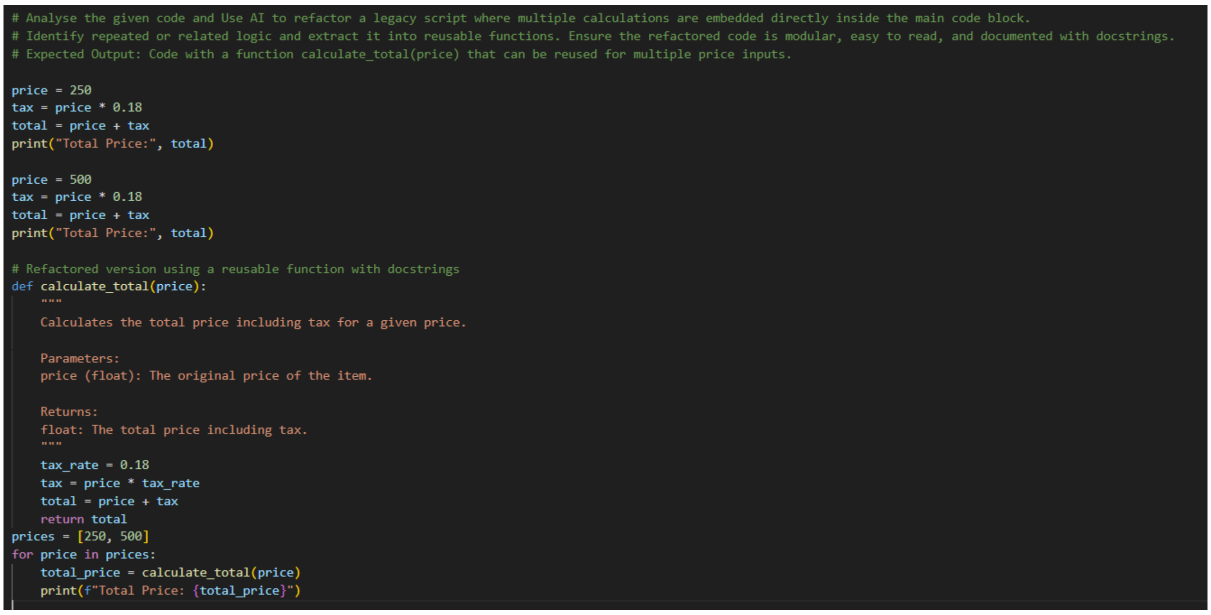
Output :



```
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab 11>
```

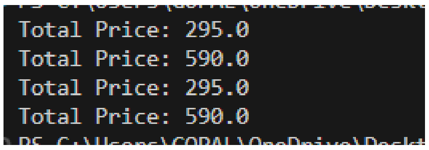
Task-2 :

Prompt : Analyse the given code and Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block. Identify repeated or related logic and extract it into reusable functions. Ensure the refactored code is modular, easy to read, and documented with docstrings. Expected Output: Code with a function calculate_total(price) that can be reused for multiple price inputs.



```python
# Analyse the given code and Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block.
# Identify repeated or related logic and extract it into reusable functions. Ensure the refactored code is modular, easy to read, and documented with docstrings.
# Expected Output: Code with a function calculate_total(price) that can be reused for multiple price inputs.

price = 250
tax = price * 0.18
total = price + tax
print("Total Price:", total)

price = 500
tax = price * 0.18
total = price + tax
print("Total Price:", total)

# Refactored version using a reusable function with docstrings
def calculate_total(price):
    """
    Calculates the total price including tax for a given price.

    Parameters:
    price (float): The original price of the item.

    Returns:
    float: The total price including tax.
    """
    tax_rate = 0.18
    tax = price * tax_rate
    total = price + tax
    return total
prices = [250, 500]
for price in prices:
    total_price = calculate_total(price)
    print(f"Total Price: {total_price}")
```

Output :



```
Total Price: 295.0
Total Price: 590.0
Total Price: 295.0
Total Price: 590.0
```

Task-3 :

Prompt : Analyse the given code and Refactor a Python script that contains repeated if–elif–else grading logic by implementing a structured, object-oriented solution using a class and a method.

Mandatory Implementation Requirements

1. Class Name: GradeCalculator
2. Method Name: calculate_grade(self, marks)
3. The method must:

Accept marks as a parameter.

Return the corresponding grade as a string.

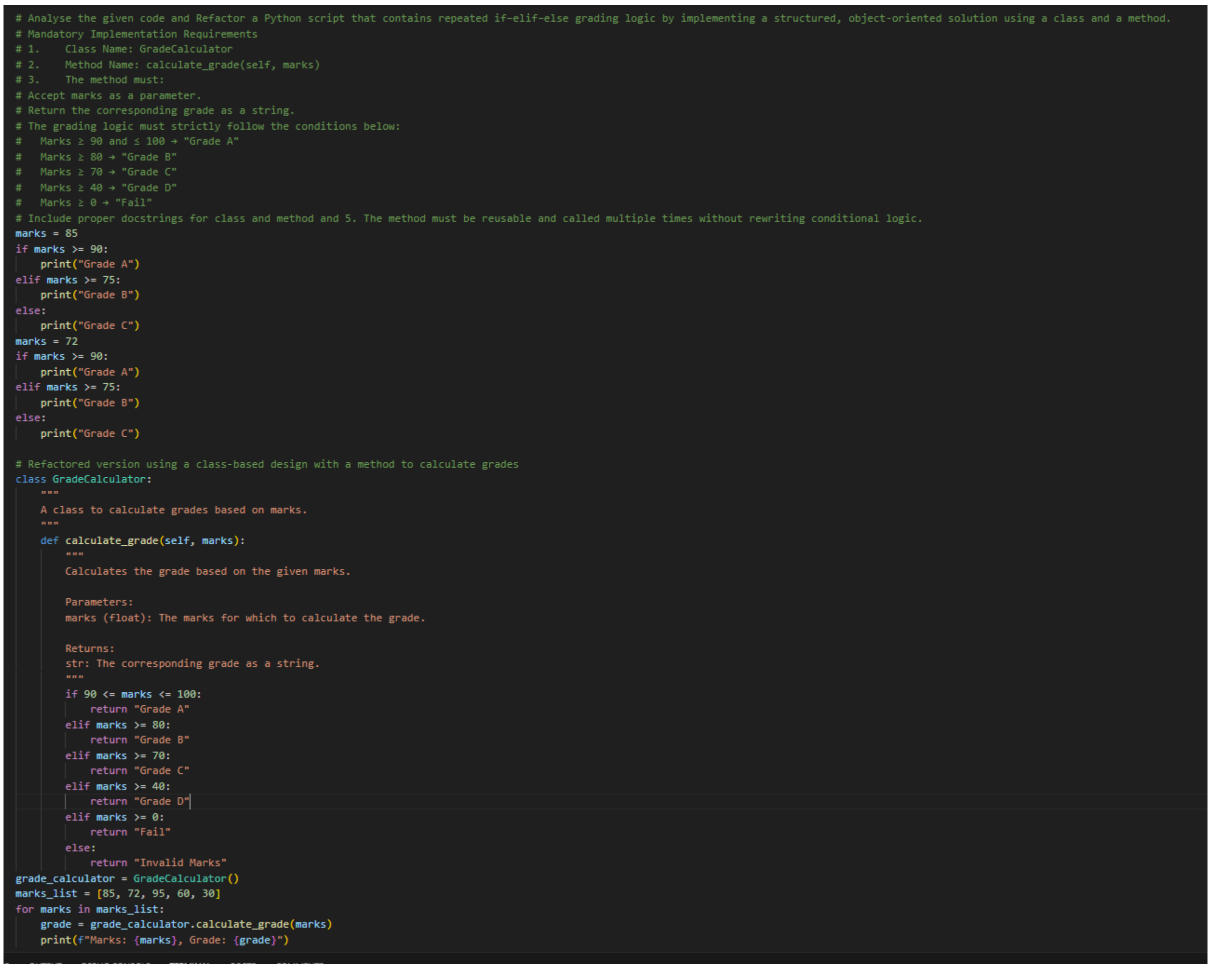
The grading logic must strictly follow the conditions below:

Marks ≥ 90 and ≤ 100 → "Grade A"

Marks ≥ 80 → "Grade B"

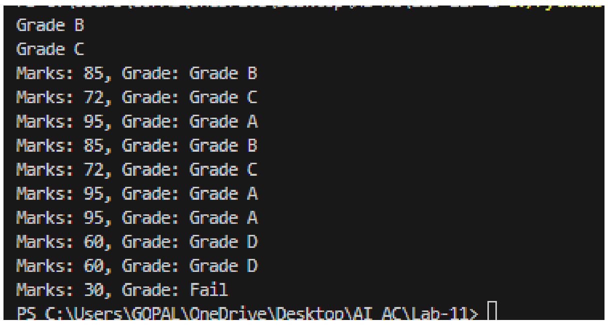
Marks ≥ 70 → "Grade C"

Marks ≥ 40 → "Grade D"
Marks ≥ 0 → "Fail"
Include proper docstrings for class and method and 5. The method must be reusable and called multiple times without rewriting conditional logic.



```python
# Analyse the given code and Refactor a Python script that contains repeated if-elif-else grading logic by implementing a structured, object-oriented solution using a class and a method.
# Mandatory Implementation Requirements
# 1.    Class Name: GradeCalculator
# 2.    Method Name: calculate_grade(self, marks)
# 3.    The method must:
# Accept marks as a parameter.
# Return the corresponding grade as a string.
# The grading logic must strictly follow the conditions below:
#   Marks ≥ 90 and ≤ 100 → "Grade A"
#   Marks ≥ 80 → "Grade B"
#   Marks ≥ 70 → "Grade C"
#   Marks ≥ 40 → "Grade D"
#   Marks ≥ 0 → "Fail"
# Include proper docstrings for class and method and 5. The method must be reusable and called multiple times without rewriting conditional logic.
marks = 85
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
marks = 72
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")

# Refactored version using a class-based design with a method to calculate grades
class GradeCalculator:
    """
    A class to calculate grades based on marks.
    """
    def calculate_grade(self, marks):
        """
        Calculates the grade based on the given marks.

        Parameters:
        marks (float): The marks for which to calculate the grade.

        Returns:
        str: The corresponding grade as a string.
        """
        if 90 <= marks <= 100:
            return "Grade A"
        elif marks >= 80:
            return "Grade B"
        elif marks >= 70:
            return "Grade C"
        elif marks >= 40:
            return "Grade D"
        elif marks >= 0:
            return "Fail"
        else:
            return "Invalid Marks"
grade_calculator = GradeCalculator()
marks_list = [85, 72, 95, 60, 30]
for marks in marks_list:
    grade = grade_calculator.calculate_grade(marks)
    print(f"Marks: {marks}, Grade: {grade}")
```

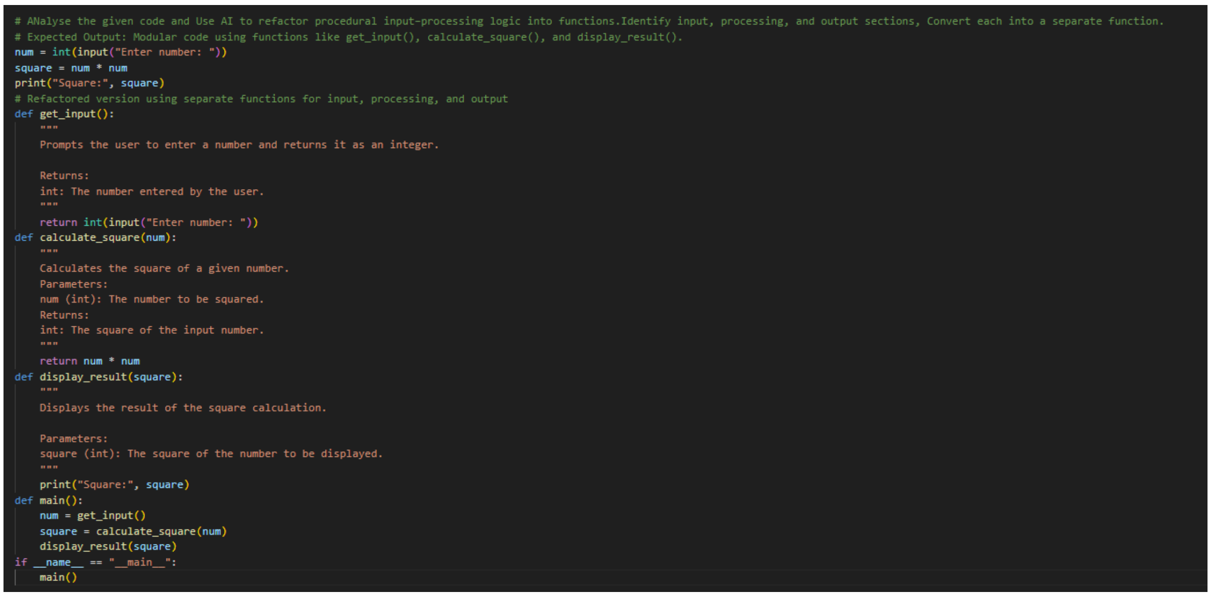
Output :



```
Grade B
Grade C
Marks: 85, Grade: Grade B
Marks: 72, Grade: Grade C
Marks: 95, Grade: Grade A
Marks: 85, Grade: Grade B
Marks: 72, Grade: Grade C
Marks: 95, Grade: Grade A
Marks: 95, Grade: Grade A
Marks: 60, Grade: Grade D
Marks: 60, Grade: Grade D
Marks: 30, Grade: Fail
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11>
```

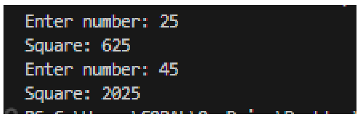
Task-4 :

Prompt : Analyse the given code and Use AI to refactor procedural input–processing logic into functions.Identify input, processing, and output sections, Convert each into a separate function. Expected Output: Modular code using functions like get_input(), calculate_square(), and display_result().



```
# ANalyse the given code and Use AI to refactor procedural input-processing logic into functions.Identify input, processing, and output sections, Convert each into a separate function.
# Expected Output: Modular code using functions like get_input(), calculate_square(), and display_result().
num = int(input("Enter number: "))
square = num * num
print("Square:", square)
# Refactored version using separate functions for input, processing, and output
def get_input():
    """
    Prompts the user to enter a number and returns it as an integer.

    Returns:
    int: The number entered by the user.
    """
    return int(input("Enter number: "))
def calculate_square(num):
    """
    Calculates the square of a given number.
    Parameters:
    num (int): The number to be squared.
    Returns:
    int: The square of the input number.
    """
    return num * num
def display_result(square):
    """
    Displays the result of the square calculation.

    Parameters:
    square (int): The square of the number to be displayed.
    """
    print("Square:", square)
def main():
    num = get_input()
    square = calculate_square(num)
    display_result(square)
if __name__ == "__main__":
    main()
```

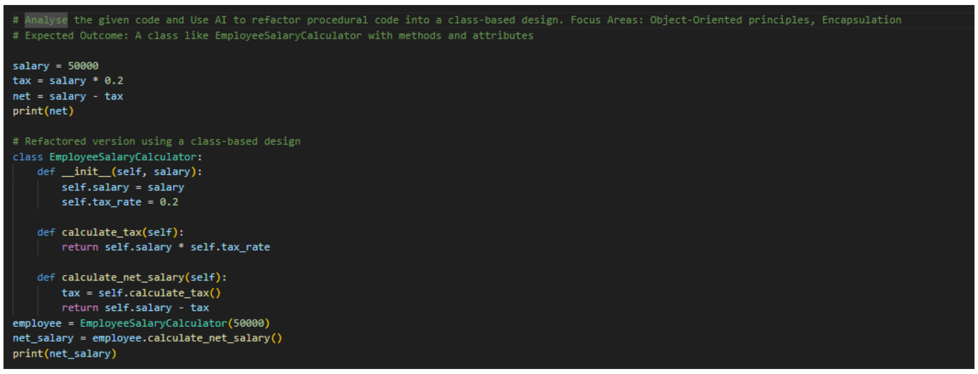
Output :



```
Enter number: 25
Square: 625
Enter number: 45
Square: 2025
```

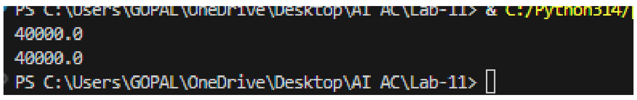
Task-5 :

Prompt : Analyse the given code and Use AI to refactor procedural code into a class-based design. Focus Areas: Object-Oriented principles, Encapsulation. Expected Outcome: A class like EmployeeSalaryCalculator with methods and attributes



```
# Analyse the given code and Use AI to refactor procedural code into a class-based design. Focus Areas: Object-Oriented principles, Encapsulation
# Expected Outcome: A class like EmployeeSalaryCalculator with methods and attributes

salary = 50000
tax = salary * 0.2
net = salary - tax
print(net)

# Refactored version using a class-based design
class EmployeeSalaryCalculator:
    def __init__(self, salary):
        self.salary = salary
        self.tax_rate = 0.2

    def calculate_tax(self):
        return self.salary * self.tax_rate

    def calculate_net_salary(self):
        tax = self.calculate_tax()
        return self.salary - tax
employee = EmployeeSalaryCalculator(50000)
net_salary = employee.calculate_net_salary()
print(net_salary)
```

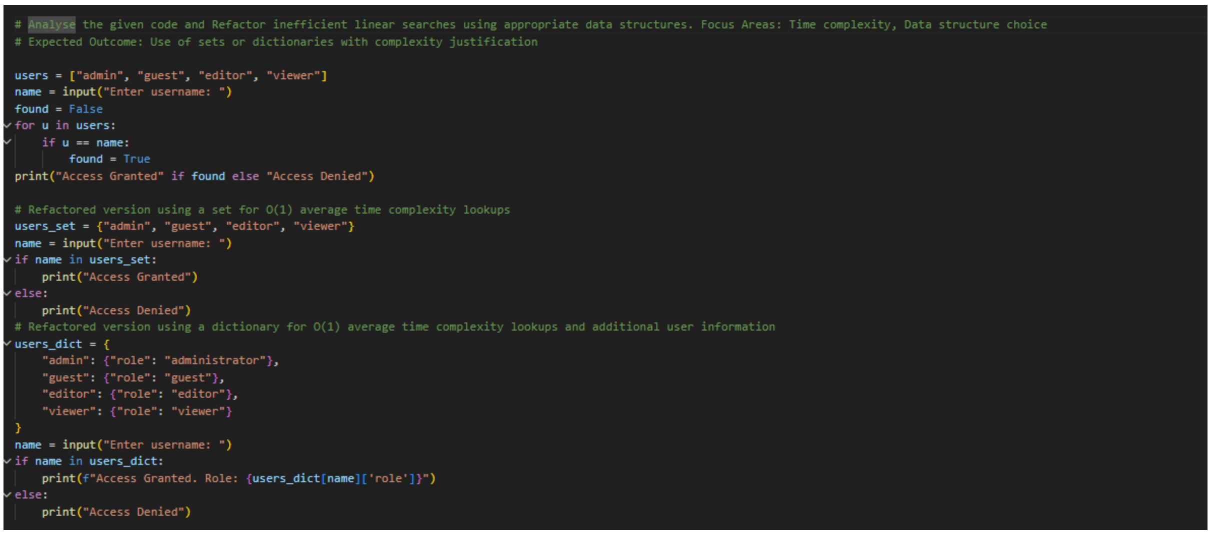
Output :



```
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> & C:/Python314/
40000.0
40000.0
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> 
```

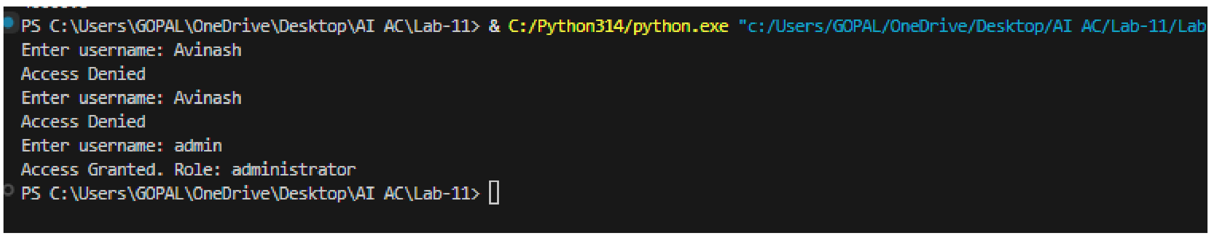
Task-6 :

Prompt : Analyse the given code and Refactor inefficient linear searches using appropriate data structures. Focus Areas: Time complexity, Data structure choice. Expected Outcome: Use of sets or dictionaries with complexity justification



```python
# Analyse the given code and Refactor inefficient linear searches using appropriate data structures. Focus Areas: Time complexity, Data structure choice
# Expected Outcome: Use of sets or dictionaries with complexity justification

users = ["admin", "guest", "editor", "viewer"]
name = input("Enter username: ")
found = False
for u in users:
    if u == name:
        found = True
print("Access Granted" if found else "Access Denied")

# Refactored version using a set for O(1) average time complexity lookups
users_set = {"admin", "guest", "editor", "viewer"}
name = input("Enter username: ")
if name in users_set:
    print("Access Granted")
else:
    print("Access Denied")
# Refactored version using a dictionary for O(1) average time complexity lookups and additional user information
users_dict = {
    "admin": {"role": "administrator"},
    "guest": {"role": "guest"},
    "editor": {"role": "editor"},
    "viewer": {"role": "viewer"}
}
name = input("Enter username: ")
if name in users_dict:
    print(f"Access Granted. Role: {users_dict[name]['role']}")
else:
    print("Access Denied")
```

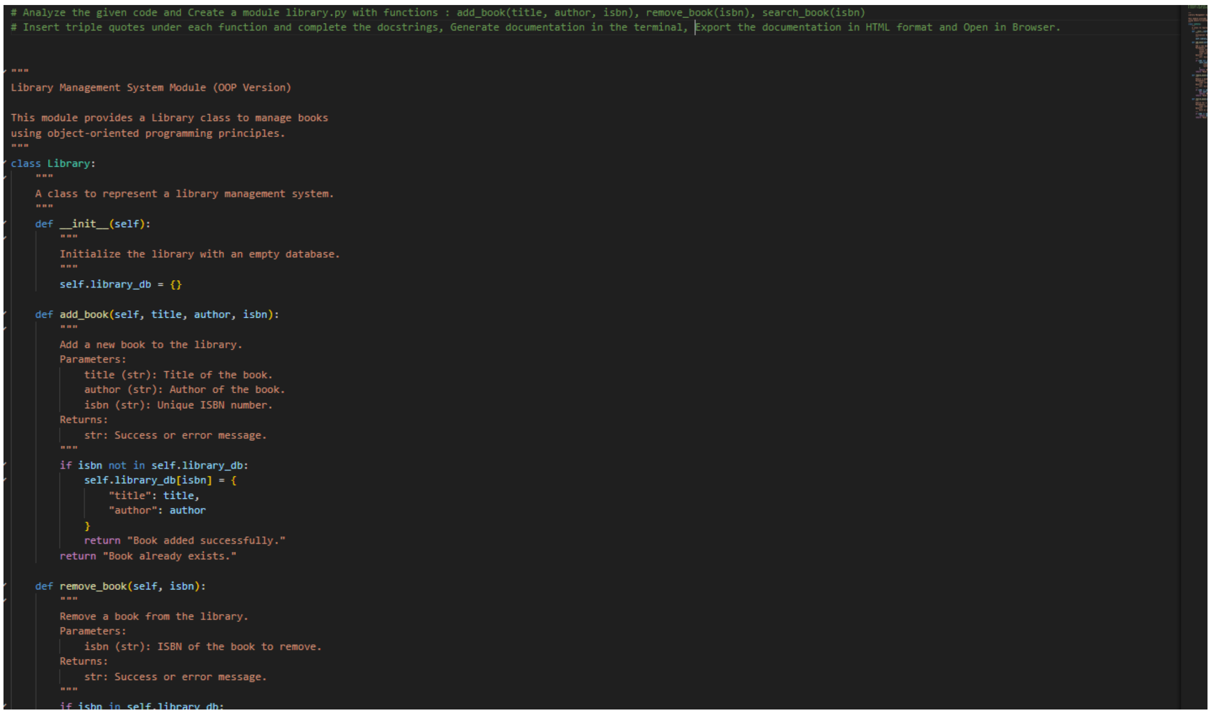
Output :



```
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> & C:/Python314/python.exe "c:/Users/GOPAL/OneDrive/Desktop/AI AC/Lab-11/Lab
Enter username: Avinash
Access Denied
Enter username: Avinash
Access Denied
Enter username: admin
Access Granted. Role: administrator
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> 
```

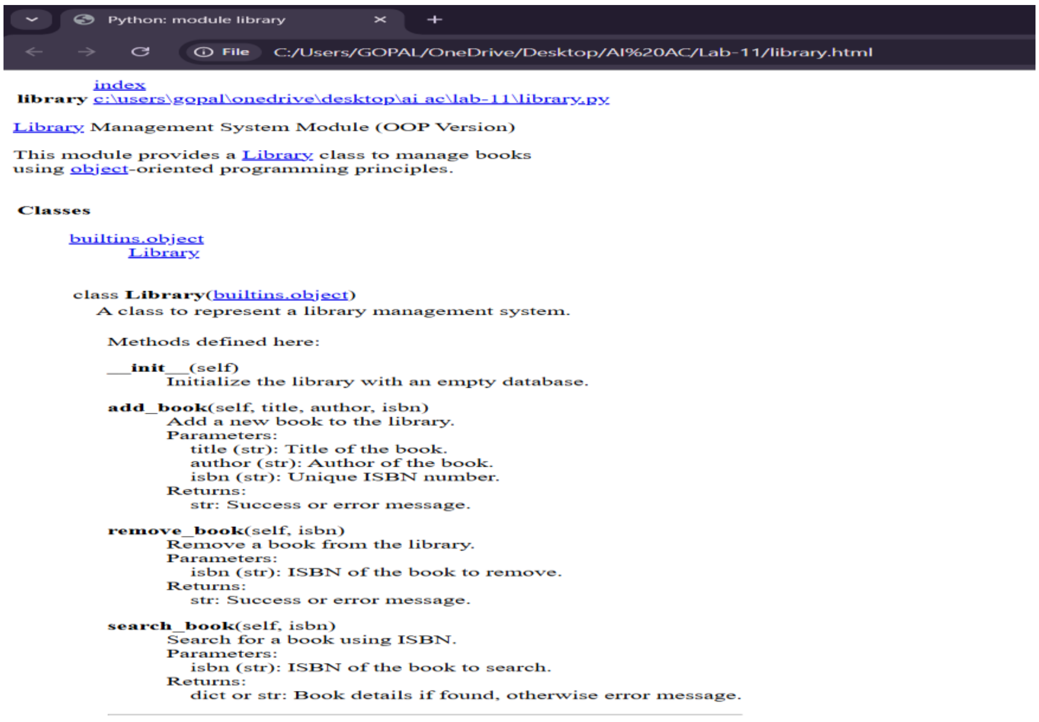
Task-7 :

Prompt : Analyze the given code and Create a module library.py with functions: add_book(title, author, isbn), remove_book(isbn), search_book(isbn). Insert triple quotes under each function and complete the docstrings, Generate documentation in the terminal, Export the documentation in HTML format and Open in Browser.



```python
# Analyze the given code and Create a module library.py with functions : add_book(title, author, isbn), remove_book(isbn), search_book(isbn)
# Insert triple quotes under each function and complete the docstrings, Generate documentation in the terminal, Export the documentation in HTML format and Open in Browser.


"""
Library Management System Module (OOP Version)

This module provides a Library class to manage books
using object-oriented programming principles.
"""
class Library:
    """
    A class to represent a library management system.
    """
    def __init__(self):
        """
        Initialize the library with an empty database.
        """
        self.library_db = {}

    def add_book(self, title, author, isbn):
        """
        Add a new book to the library.
        Parameters:
            title (str): Title of the book.
            author (str): Author of the book.
            isbn (str): Unique ISBN number.
        Returns:
            str: Success or error message.
        """
        if isbn not in self.library_db:
            self.library_db[isbn] = {
                "title": title,
                "author": author
            }
            return "Book added successfully."
        return "Book already exists."

    def remove_book(self, isbn):
        """
        Remove a book from the library.
        Parameters:
            isbn (str): ISBN of the book to remove.
        Returns:
            str: Success or error message.
        """
        if isbn in self.library_db:
```

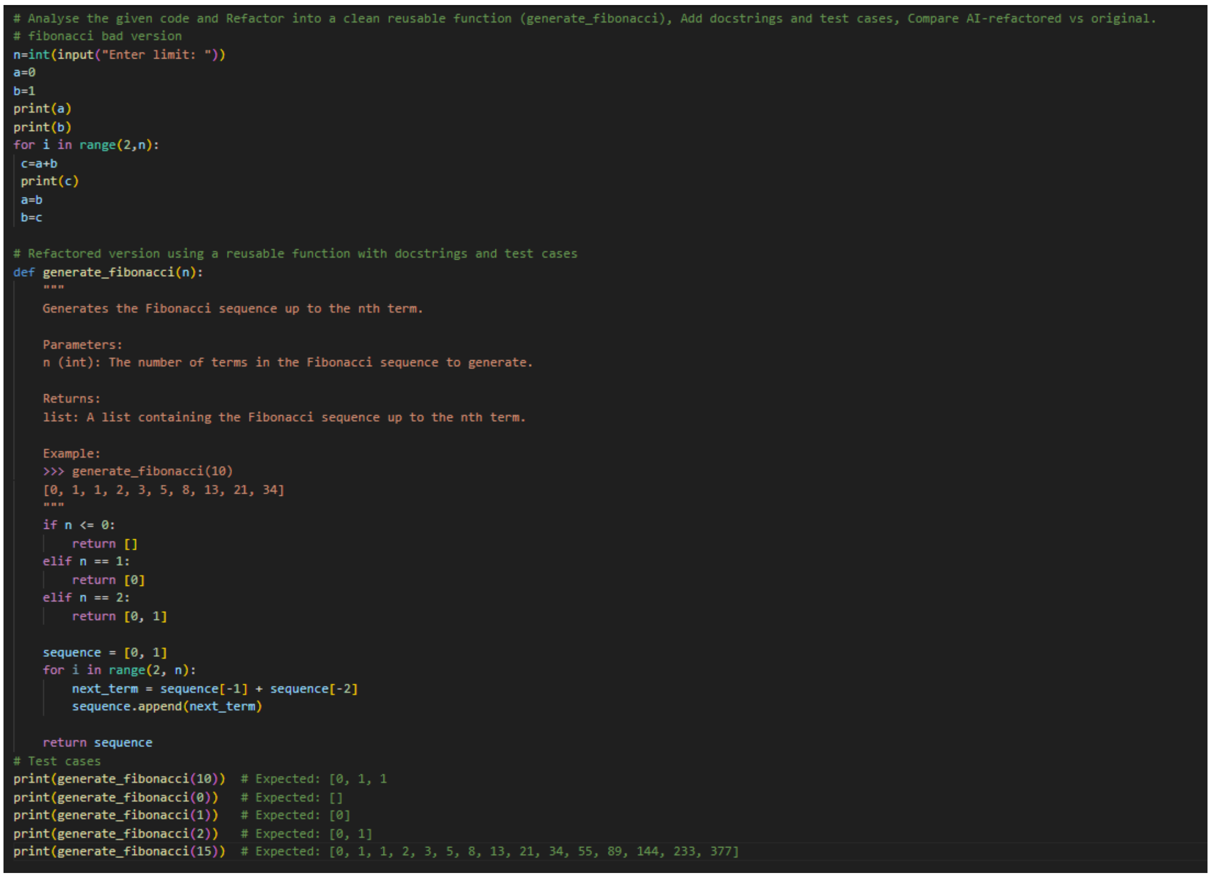
Output :



```
                                                     index
library c:\users\gopal\onedrive\desktop\ai ac\lab-11\library.py

Library Management System Module (OOP Version)

This module provides a Library class to manage books
using object-oriented programming principles.

 Classes

        builtins.object
                Library

        class Library(builtins.object)
           A class to represent a library management system.

             Methods defined here:

             __init__(self)
                     Initialize the library with an empty database.

             add_book(self, title, author, isbn)
                     Add a new book to the library.
                     Parameters:
                         title (str): Title of the book.
                         author (str): Author of the book.
                         isbn (str): Unique ISBN number.
                     Returns:
                         str: Success or error message.

             remove_book(self, isbn)
                     Remove a book from the library.
                     Parameters:
                         isbn (str): ISBN of the book to remove.
                     Returns:
                         str: Success or error message.

             search_book(self, isbn)
                     Search for a book using ISBN.
                     Parameters:
                         isbn (str): ISBN of the book to search.
                     Returns:
                         dict or str: Book details if found, otherwise error message.
```

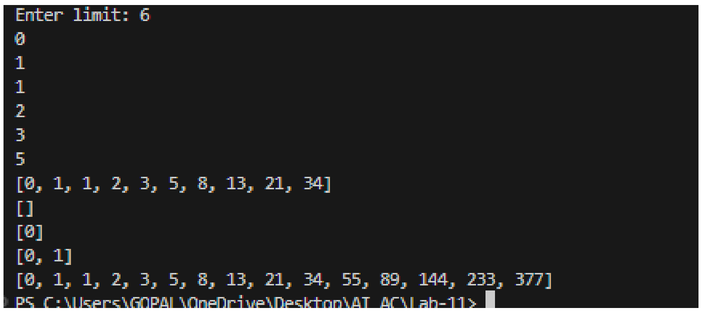
Task-8 :

Prompt : Analyse the given code and Refactor into a clean reusable function (generate_fibonacci), Add docstrings and test cases, Compare AI-refactored vs original.



```python
# Analyse the given code and Refactor into a clean reusable function (generate_fibonacci), Add docstrings and test cases, Compare AI-refactored vs original.
# fibonacci bad version
n=int(input("Enter limit: "))
a=0
b=1
print(a)
print(b)
for i in range(2,n):
 c=a+b
 print(c)
 a=b
 b=c

# Refactored version using a reusable function with docstrings and test cases
def generate_fibonacci(n):
    """
    Generates the Fibonacci sequence up to the nth term.

    Parameters:
    n (int): The number of terms in the Fibonacci sequence to generate.

    Returns:
    list: A list containing the Fibonacci sequence up to the nth term.

    Example:
    >>> generate_fibonacci(10)
    [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
    """
    if n <= 0:
        return []
    elif n == 1:
        return [0]
    elif n == 2:
        return [0, 1]

    sequence = [0, 1]
    for i in range(2, n):
        next_term = sequence[-1] + sequence[-2]
        sequence.append(next_term)

    return sequence
# Test cases
print(generate_fibonacci(10))  # Expected: [0, 1, 1
print(generate_fibonacci(0))   # Expected: []
print(generate_fibonacci(1))   # Expected: [0]
print(generate_fibonacci(2))   # Expected: [0, 1]
print(generate_fibonacci(15))  # Expected: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
```

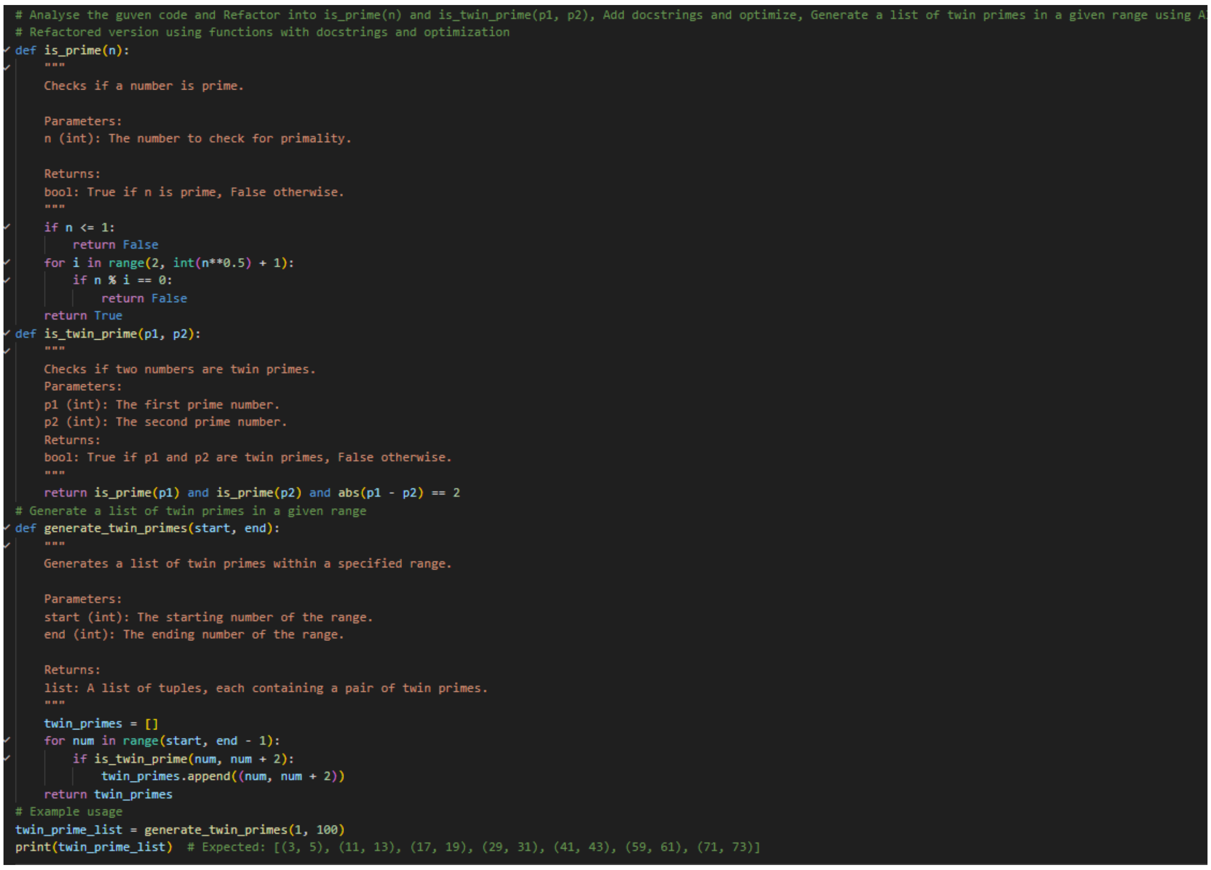
Output :



```
Enter limit: 6
0
1
1
2
3
5
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
[]
[0]
[0, 1]
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377]
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11>
```

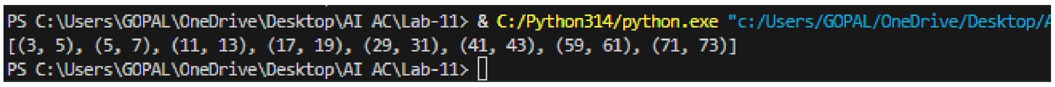
Task-9 :

Prompt : Analyse the guven code and Refactor into is_prime(n) and is_twin_prime(p1, p2), Add docstrings and optimize, Generate a list of twin primes in a given range using AI



```python
# Analyse the guven code and Refactor into is_prime(n) and is_twin_prime(p1, p2), Add docstrings and optimize, Generate a list of twin primes in a given range using A
# Refactored version using functions with docstrings and optimization
def is_prime(n):
    """
    Checks if a number is prime.

    Parameters:
    n (int): The number to check for primality.

    Returns:
    bool: True if n is prime, False otherwise.
    """
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
def is_twin_prime(p1, p2):
    """
    Checks if two numbers are twin primes.
    Parameters:
    p1 (int): The first prime number.
    p2 (int): The second prime number.
    Returns:
    bool: True if p1 and p2 are twin primes, False otherwise.
    """
    return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
# Generate a list of twin primes in a given range
def generate_twin_primes(start, end):
    """
    Generates a list of twin primes within a specified range.

    Parameters:
    start (int): The starting number of the range.
    end (int): The ending number of the range.

    Returns:
    list: A list of tuples, each containing a pair of twin primes.
    """
    twin_primes = []
    for num in range(start, end - 1):
        if is_twin_prime(num, num + 2):
            twin_primes.append((num, num + 2))
    return twin_primes
# Example usage
twin_prime_list = generate_twin_primes(1, 100)
print(twin_prime_list)  # Expected: [(3, 5), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
```

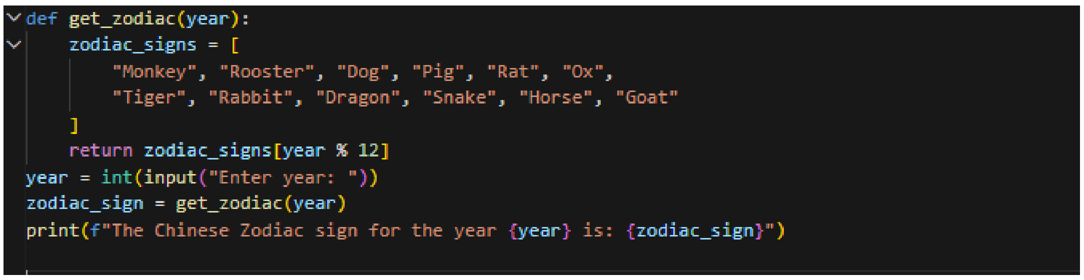
Output :



```
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> & C:/Python314/python.exe "c:/Users/GOPAL/OneDrive/Desktop/A
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
PS C:\Users\GOPAL\OneDrive\Desktop\AI AC\Lab-11> 
```

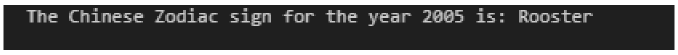
Task-10 :

Prompt : Refactor the code to determine the Chinese Zodiac sign based on the year input using function get_zodiac(year) and return the zodiac sign instead of printing it directly.



```python
def get_zodiac(year):
    zodiac_signs = [
        "Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox",
        "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Goat"
    ]
    return zodiac_signs[year % 12]
year = int(input("Enter year: "))
zodiac_sign = get_zodiac(year)
print(f"The Chinese Zodiac sign for the year {year} is: {zodiac_sign}")
```

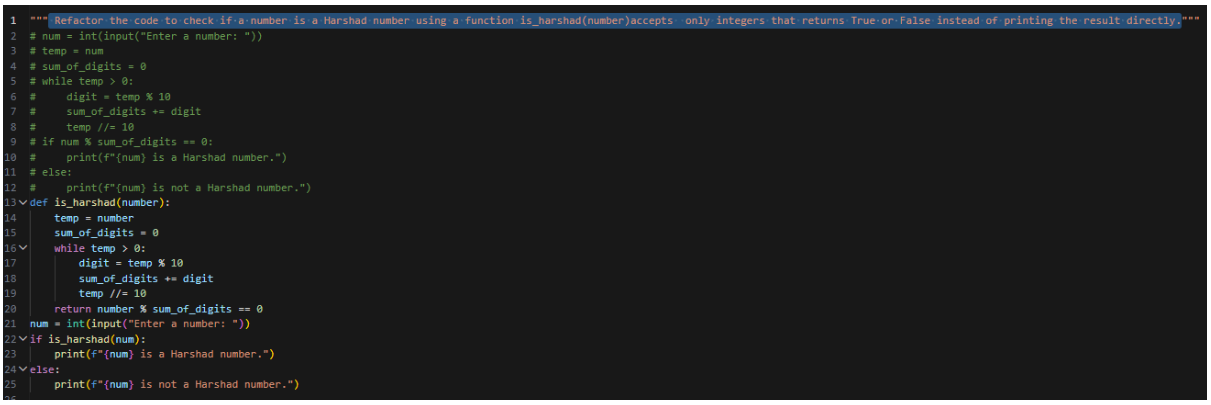
Output :



```
The Chinese Zodiac sign for the year 2005 is: Rooster
```

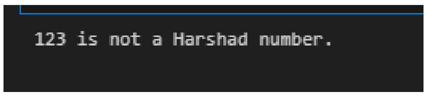
Task-11 :

Prompt : Refactor the code to check if a number is a Harshad number using a function is_harshad(number)accepts only integers that returns True or False instead of printing the result directly.



```python
""" Refactor the code to check if a number is a Harshad number using a function is_harshad(number)accepts  only integers that returns True or False instead of printing the result directly."""
# num = int(input("Enter a number: "))
# temp = num
# sum_of_digits = 0
# while temp > 0:
#     digit = temp % 10
#     sum_of_digits += digit
#     temp //= 10
# if num % sum_of_digits == 0:
#     print(f"{num} is a Harshad number.")
# else:
#     print(f"{num} is not a Harshad number.")
def is_harshad(number):
    temp = number
    sum_of_digits = 0
    while temp > 0:
        digit = temp % 10
        sum_of_digits += digit
        temp //= 10
    return number % sum_of_digits == 0
num = int(input("Enter a number: "))
if is_harshad(num):
    print(f"{num} is a Harshad number.")
else:
    print(f"{num} is not a Harshad number.")
```

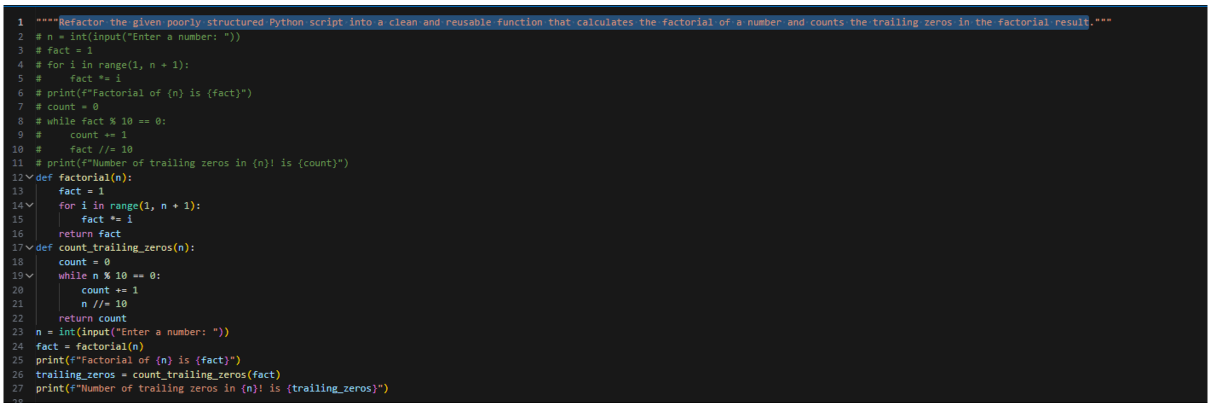
Output :



```
123 is not a Harshad number.
```

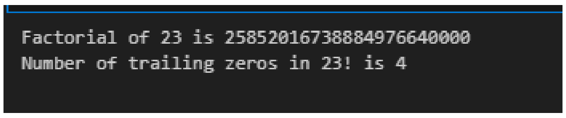
Task-12 :

Prompt : Refactor the given poorly structured Python script into a clean and reusable function that calculates the factorial of a number and counts the trailing zeros in the factorial result.



```python
""""Refactor the given poorly structured Python script into a clean and reusable function that calculates the factorial of a number and counts the trailing zeros in the factorial result."""
# n = int(input("Enter a number: "))
# fact = 1
# for i in range(1, n + 1):
#     fact *= i
# print(f"Factorial of {n} is {fact}")
# count = 0
# while fact % 10 == 0:
#     count += 1
#     fact //= 10
# print(f"Number of trailing zeros in {n}! is {count}")
def factorial(n):
    fact = 1
    for i in range(1, n + 1):
        fact *= i
    return fact
def count_trailing_zeros(n):
    count = 0
    while n % 10 == 0:
        count += 1
        n //= 10
    return count
n = int(input("Enter a number: "))
fact = factorial(n)
print(f"Factorial of {n} is {fact}")
trailing_zeros = count_trailing_zeros(fact)
print(f"Number of trailing zeros in {n}! is {trailing_zeros}")
```

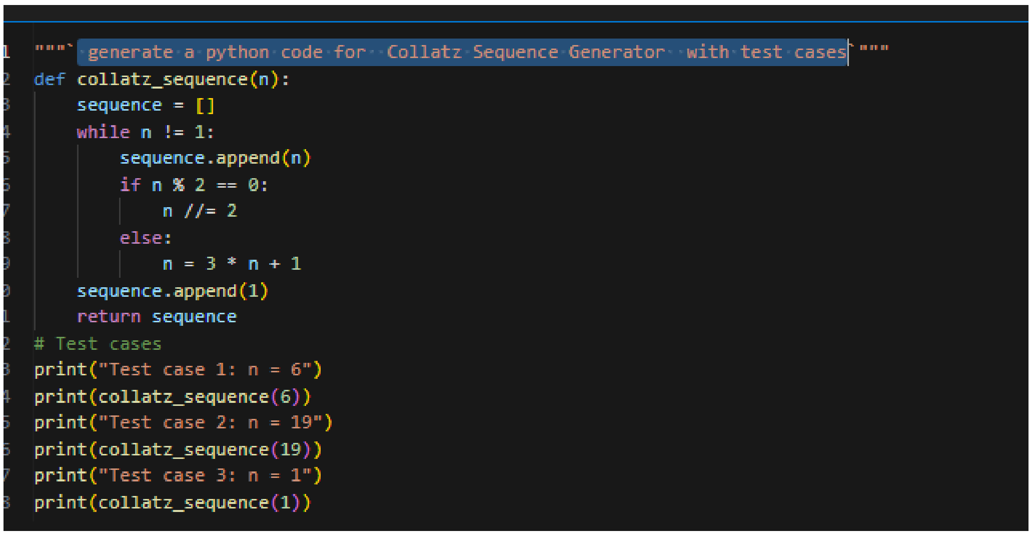
Output :



```
Factorial of 23 is 25852016738884976640000
Number of trailing zeros in 23! is 4
```

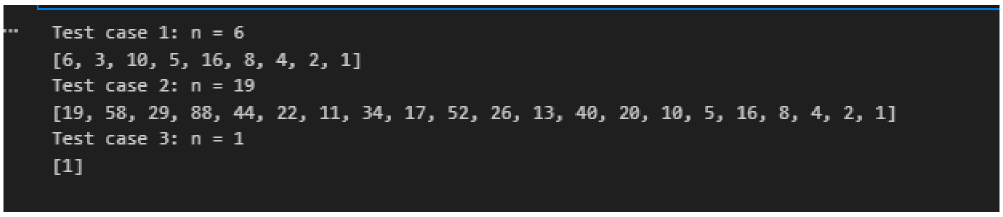
Task-13 :

Prompt : generate a python code for Collatz Sequence Generator with test cases



```python
""" generate a python code for  Collatz Sequence Generator  with test cases"""
def collatz_sequence(n):
    sequence = []
    while n != 1:
        sequence.append(n)
        if n % 2 == 0:
            n //= 2
        else:
            n = 3 * n + 1
    sequence.append(1)
    return sequence
# Test cases
print("Test case 1: n = 6")
print(collatz_sequence(6))
print("Test case 2: n = 19")
print(collatz_sequence(19))
print("Test case 3: n = 1")
print(collatz_sequence(1))
```

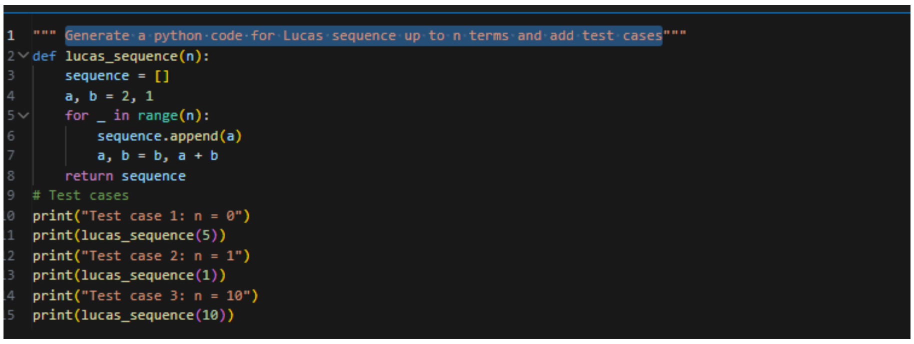
Output :



```
Test case 1: n = 6
[6, 3, 10, 5, 16, 8, 4, 2, 1]
Test case 2: n = 19
[19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
Test case 3: n = 1
[1]
```

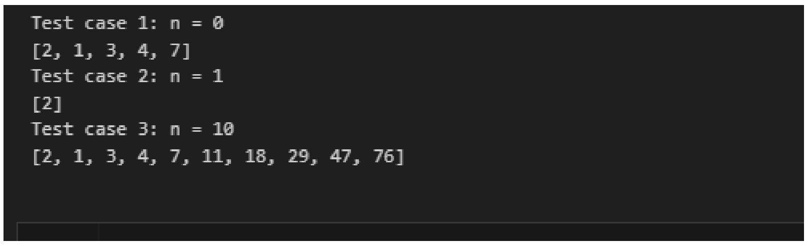
Task-14 :

Prompt : Generate a python code for Lucas sequence up to n terms and add test cases.



```python
""" Generate a python code for Lucas sequence up to n terms and add test cases"""
def lucas_sequence(n):
    sequence = []
    a, b = 2, 1
    for _ in range(n):
        sequence.append(a)
        a, b = b, a + b
    return sequence
# Test cases
print("Test case 1: n = 0")
print(lucas_sequence(5))
print("Test case 2: n = 1")
print(lucas_sequence(1))
print("Test case 3: n = 10")
print(lucas_sequence(10))
```

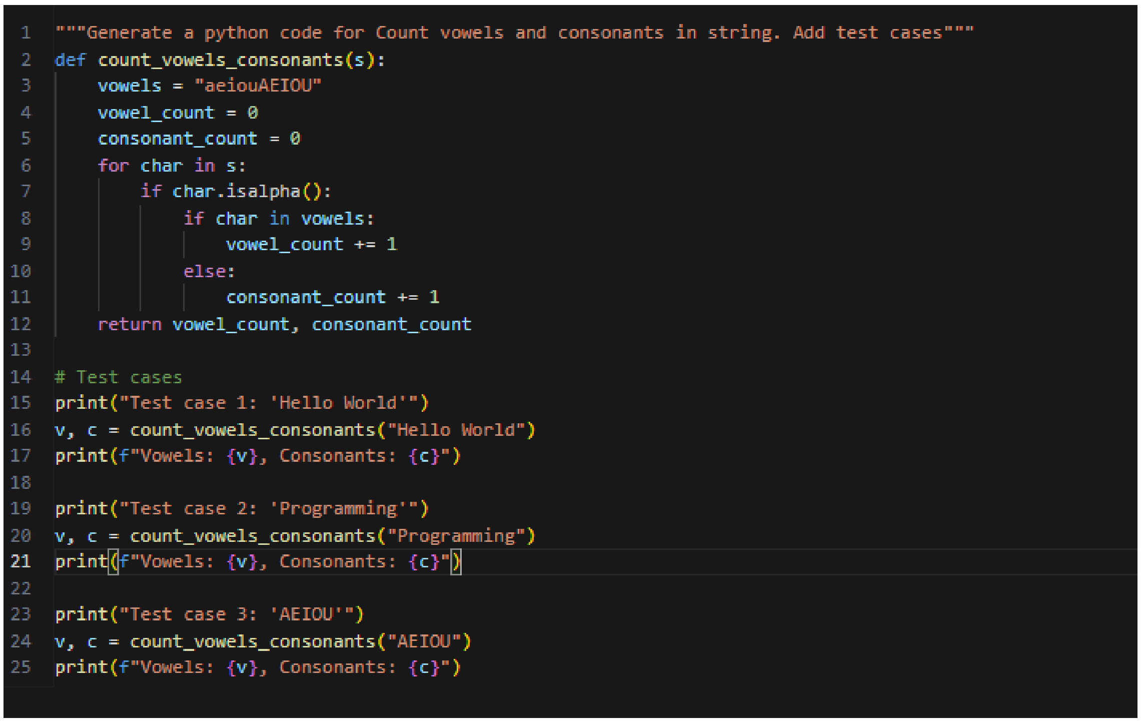
Output :



```
Test case 1: n = 0
[2, 1, 3, 4, 7]
Test case 2: n = 1
[2]
Test case 3: n = 10
[2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
```

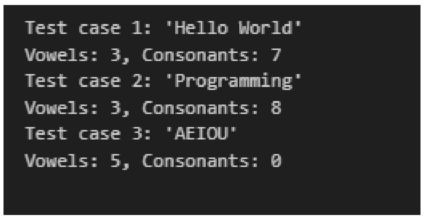
Task-15 :

Prompt : Generate a python code for Count vowels and consonants in string. Add test cases



```python
"""Generate a python code for Count vowels and consonants in string. Add test cases"""
def count_vowels_consonants(s):
    vowels = "aeiouAEIOU"
    vowel_count = 0
    consonant_count = 0
    for char in s:
        if char.isalpha():
            if char in vowels:
                vowel_count += 1
            else:
                consonant_count += 1
    return vowel_count, consonant_count

# Test cases
print("Test case 1: 'Hello World'")
v, c = count_vowels_consonants("Hello World")
print(f"Vowels: {v}, Consonants: {c}")

print("Test case 2: 'Programming'")
v, c = count_vowels_consonants("Programming")
print(f"Vowels: {v}, Consonants: {c}")

print("Test case 3: 'AEIOU'")
v, c = count_vowels_consonants("AEIOU")
print(f"Vowels: {v}, Consonants: {c}")
```

Output :



```
Test case 1: 'Hello World'
Vowels: 3, Consonants: 7
Test case 2: 'Programming'
Vowels: 3, Consonants: 8
Test case 3: 'AEIOU'
Vowels: 5, Consonants: 0
```