Assignment-13

2303A510G6

G.Charanya

Batch-23

Task Description #1 (Refactoring – Removing Code Duplication)

• Task: Use AI to refactor a given Python script that contains multiple repeated code blocks.

• Instructions:

o Prompt AI to identify duplicate logic and replace it with functions or classes.

o Ensure the refactored code maintains the same output.

o Add docstrings to all functions.

• Sample Legacy Code:

```
# Legacy script with repeated logic
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))
```

• Expected Output:

o Refactored code with a reusable function and no duplication.

o Well documented code

```python
#task-1
def calculate_rectangle_properties(length, width):
    """
    Calculate and print the area and perimeter of a rectangle.
    Parameters:
    length (int or float): The length of the rectangle.
    width (int or float): The width of the rectangle.
    Returns:
    None: This function prints the area and perimeter.
    """
    area = length * width
    perimeter = 2 * (length + width)
    print("Area of Rectangle:", area)
    print("Perimeter of Rectangle:", perimeter)
def main():
    """
    Main function to execute rectangle calculations
    for multiple sets of dimensions.

    Returns:
    None
    """
    rectangles = [(5, 10), (7, 12), (10, 15)]
    for length, width in rectangles:
        calculate_rectangle_properties(length, width)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Area of Rectangle: 50
Perimeter of Rectangle: 30
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task Description #2 (Refactoring – Extracting Reusable Functions)

• Task: Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block.

• Instructions:

o Identify repeated or related logic and extract it into reusable functions.

o Ensure the refactored code is modular, easy to read, and

documented with docstrings.

• Sample Legacy Code:

Week7

-

Monda

y

```
# Legacy script with inline repeated logic

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)
```

• Expected Output:

o Code with a function calculate_total(price) that can be reused

for multiple price inputs.

o Well documented code

```python
#task-2
def calculate_total(price):
    """
    Calculate the total price including 18% tax.
    Parameters:
    price (int or float): The original price of the product
    Returns:
    float: The total price after adding 18% tax.
    """
    tax = price * 0.18
    total = price + tax
    return total
def main():
    """
    Main function to calculate and print total prices
    for multiple product values.
    Returns:
    None
    """
    prices = [250, 500]
    for price in prices:
        total = calculate_total(price)
        print("Total Price:", total)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Total Price: 295.0
Total Price: 590.0
```

Task Description #3: Refactoring Using Classes and Methods (Eliminating Redundant Conditional Logic)

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.

Problem Statement

The given script contains duplicated conditional statements used to assign grades based on student marks. This redundancy violates clean code principles and reduces maintainability.

You are required to refactor the script using a class-based design to improve modularity, reusability, and readability while preserving the original grading logic.

Mandatory Implementation Requirements

1. Class Name: GradeCalculator

2. Method Name: calculate_grade(self, marks)

3. The method must:

o Accept marks as a parameter.

o Return the corresponding grade as a string.

o 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"

Note: Assume marks are within the valid range of 0 to 100.

4. Include proper docstrings for:

o The class

o The method (with parameter and return descriptions)

5. The method must be reusable and called multiple times without rewriting conditional logic.

• Given code:

```
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")
```

Expected Output:

- Define a class named GradeCalculator.
- Implement a method calculate_grade(self, marks) inside the class.
- Create an object of the class.
- Call the method for different student marks.

• Print the returned grade values.

ass_13.py > ...

```python
#task-3
class GradeCalculator:
    """
    A class used to calculate student grades based on marks.
    Methods:
    calculate_grade(self, marks):
        Returns the grade string based on the given marks.
    """
    def calculate_grade(self, marks):
        """
        Determine the grade for a given mark.
        Parameters:
        marks (int or float): The student's marks (assumed between 0 and 100).
        Returns:
        str: The grade corresponding to the marks based on the grading rules.
        """
        if marks >= 90 and 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"
def main():
    """
    Main function to demonstrate reusable grade calculation.
    """
    calculator = GradeCalculator()
    marks_list = [85, 72, 95, 38]
    for marks in marks_list:
        grade = calculator.calculate_grade(marks)
        print(f"Marks: {marks} -> {grade}")
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Marks: 85 -> Grade B
Marks: 72 -> Grade C
Marks: 95 -> Grade A
Marks: 38 -> Fail
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task Description #4 (Refactoring – Converting Procedural Code to Functions)

• Task: Use AI to refactor procedural input–processing logic into functions.

Instructions:

o Identify input, processing, and output sections.

o Convert each into a separate function.

o Improve code readability without changing behavior.

• Sample Legacy Code:

num = int(input("Enter number: "))

square = num * num

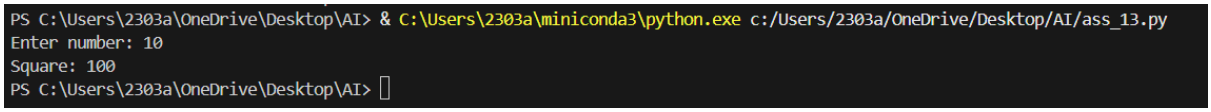
print("Square:", square)

• Expected Output:

o Modular code using functions like get_input(), calculate_square(), and display_result().

ass_13.py > ...

```python
#task-4
def get_input():
    """
    Prompt the user to enter a number.
    Returns:
    int: The number entered by the user.
    """
    return int(input("Enter number: "))
def calculate_square(num):
    """
    Calculate the square of a number.
    Parameters:
    num (int or float): The number to be squared.
    Returns:
    int or float: The square of the given number.
    """
    return num * num
def display_result(square):
    """
    Display the square result.
    Parameters:
    square (int or float): The calculated square value.
    Returns:
    None
    """
    print("Square:", square)
def main():
    """
    Main function to coordinate input, processing,
    and output operations.
    """
    number = get_input()
    square = calculate_square(number)
    display_result(square)
if __name__ == "__main__":
    main()
```



Task 5 (Refactoring Procedural Code into OOP Design)

• Task: Use AI to refactor procedural code into a class-based design.

Focus Areas:

o Object-Oriented principles

o Encapsulation

Legacy Code:

```
salary = 50000

tax = salary * 0.2

net = salary - tax

print(net)
```

Expected Outcome:

o A class like EmployeeSalaryCalculator with methods and attributes.

ass_13.py > ...

```python
#task-5
class EmployeeSalaryCalculator:
    """
    A class to calculate an employee's net salary after tax deduction.
    Attributes:
    salary (float): The employee's gross salary.
    tax_rate (float): The tax percentage applied to the salary.
    """
    def __init__(self, salary, tax_rate=0.2):
        """
        Initialize the EmployeeSalaryCalculator with salary and tax rate.
        Parameters:
        salary (float): The employee's gross salary.
        tax_rate (float, optional): The tax rate (default is 0.2 or 20%).
        """
        self.salary = salary
        self.tax_rate = tax_rate
    def calculate_tax(self):
        """
        Calculate the tax amount based on salary and tax rate.
        Returns:
        float: The calculated tax amount.
        """
        return self.salary * self.tax_rate
    def calculate_net_salary(self):
        """
        Calculate the net salary after tax deduction.
        Returns:
        float: The net salary.
        """
        tax = self.calculate_tax()
        return self.salary - tax
    def display_net_salary(self):
        """
        Display the net salary.
        Returns:
        None
        """
        net_salary = self.calculate_net_salary()
        print("Net Salary:", net_salary)
def main():
    """
    Main function to demonstrate salary calculation.
    """
    employee = EmployeeSalaryCalculator(50000)
    employee.display_net_salary()
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Net Salary: 40000.0
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 6 (Optimizing Search Logic)

• Task: Refactor inefficient linear searches using appropriate data structures.

• Focus Areas:

o Time complexity

o Data structure choice

Legacy Code:

```
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")
```

Expected Outcome:

o Use of sets or dictionaries with complexity justification

ass_13.py > ...

```python
#task-6
def check_access(users_set, username):
    """
    Check whether the given username exists in the system.
    Parameters:
    users_set (set): A set of valid usernames.
    username (str): The username to verify.
    Returns:
    bool: True if user exists, otherwise False.
    """
    return username in users_set
def main():
    """
    Main function to handle user input and access verification.
    """
    users = {"admin", "guest", "editor", "viewer"}  # Using set for O(1) lookup
    name = input("Enter username: ")
    if check_access(users, name):
        print("Access Granted")
    else:
        print("Access Denied")
if __name__ == "__main__":
    main()

```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Enter username: charanya
Access Denied
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 7 – Refactoring the Library Management System

Problem Statement

You are provided with a poorly structured Library Management script that:

• Contains repeated conditional logic

• Does not use reusable functions

• Lacks documentation

• Uses print-based procedural execution

• Does not follow modular programming principles

Your task is to refactor the code into a proper format

1. Create a module library.py with functions:

o add_book(title, author, isbn)

o remove_book(isbn)

o search_book(isbn)

2. Insert triple quotes under each function and let Copilot complete the

docstrings.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format.

5. Open the file in a browser.

Given Code

```
# Library Management System (Unstructured Version)

# This code needs refactoring into a proper module with documentation.

library_db = {}

# Adding first book

title = "Python Basics"

author = "John Doe"

isbn = "101"

if isbn not in library_db:

library_db[isbn] = {"title": title, "author": author}

print("Book added successfully.")

else:

print("Book already exists.")
```

```
# Adding second book (duplicate logic)
title = "AI Fundamentals"
author = "Jane Smith"
isbn = "102"
if isbn not in library_db:
library_db[isbn] = {"title": title, "author": author}
print("Book added successfully.")
else:
print("Book already exists.")
# Searching book (repeated logic structure)
isbn = "101"
if isbn in library_db:
print("Book Found:", library_db[isbn])
else:
print("Book not found.")
# Removing book (again repeated pattern)
isbn = "101"
if isbn in library_db:
del library_db[isbn]
print("Book removed successfully.")
else:
print("Book not found.")
```

Searching again

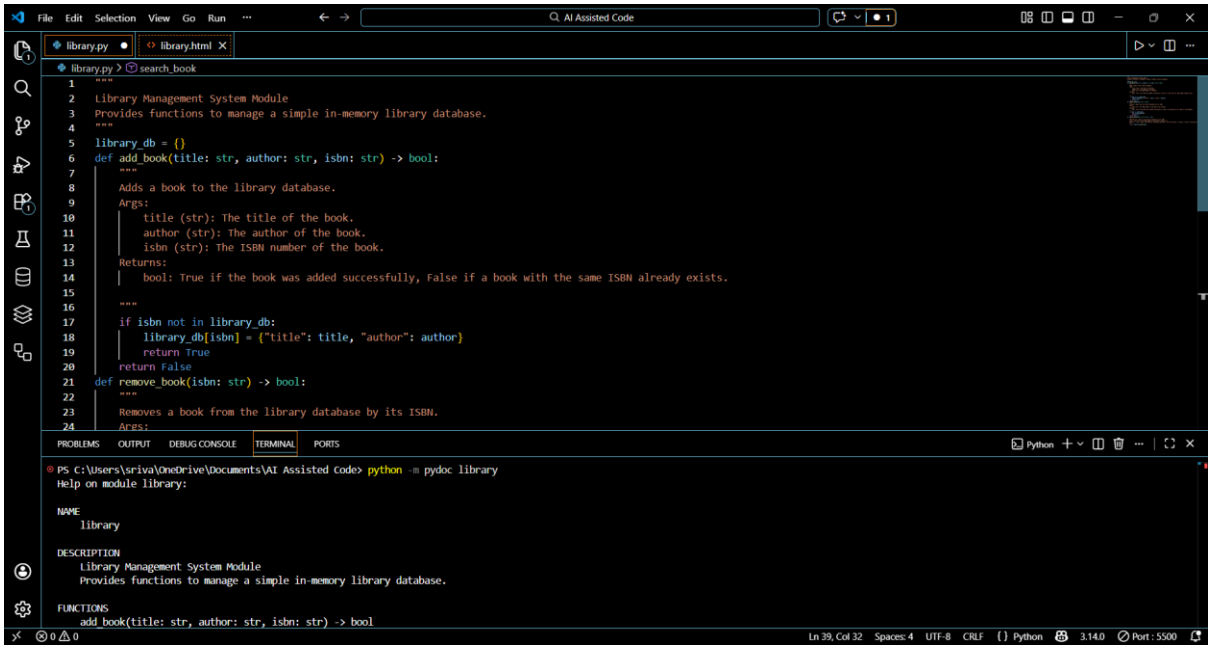
isbn = "101"

if isbn in library_db:

print("Book Found:", library_db[isbn])

else:

print("Book not found.")



index

library c:\users\sriva\onedrive\documents\ai assisted code\library.py

Library Management System Module
Provides functions to manage a simple in-memory library database.

Functions

add_book(title: str, author: str, isbn: str) -> bool
Adds a book to the library database.
Args:
title (str): The title of the book.
author (str): The author of the book.
isbn (str): The ISBN number of the book.
Returns:
bool: True if the book was added successfully, False if a book with the same ISBN already exists.

remove_book(isbn: str) -> bool
Removes a book from the library database by its ISBN.
Args:
isbn (str): The ISBN number of the book to be removed.
Returns:
bool: True if the book was removed successfully, False if the book was not found in the database.

search_book(isbn: str) -> dict | None
Searches for a book in the library database by its ISBN.
Args: isbn (str): The ISBN number of the book to search for.
Returns: dict | None: A dictionary containing the book's title and author if found, or None if the book is not found in the database.

Data

library_db = {}

Task 8– Fibonacci Generator

Write a program to generate Fibonacci series up to n.

The initial code has:

• Global variables.

• Inefficient loop.

• No functions or modularity.

Task for Students:

• Refactor into a clean reusable function (generate_fibonacci).

• Add docstrings and test cases.

• Compare AI-refactored vs original.

Bad Code Version:

```
# 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
```

ass_13.py > ...

```python
#task-8
def generate_fibonacci(n):
    """
    Generate Fibonacci series up to n terms.
    Parameters:
    n (int): Number of terms to generate. Must be >= 0.
    Returns:
    list: A list containing the Fibonacci sequence up to n terms.
    """
    if n <= 0:
        return []
    if n == 1:
        return [0]
    fibonacci_series = [0, 1]
    for _ in range(2, n):
        fibonacci_series.append(
            fibonacci_series[-1] + fibonacci_series[-2]
        )
    return fibonacci_series
def main():
    """
    Main function to demonstrate Fibonacci generation.
    """
    n = int(input("Enter limit: "))
    result = generate_fibonacci(n)
    print("Fibonacci Series:", result)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Enter limit: 10
Fibonacci Series: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 9 – Twin Primes Checker

Twin primes are pairs of primes that differ by 2 (e.g., 11 and 13, 17 and 19).

The initial code has:

• Inefficient prime checking.

• No functions.

• Hardcoded inputs.

Task for Students:

• 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.

Bad Code Version:

```
# twin primes bad version

a=11

b=13

fa=0

for i in range(2,a):

if a%i==0:

fa=1

fb=0

for i in range(2,b):

if b%i==0:

fb=1

if fa==0 and fb==0 and abs(a-b)==2:

print("Twin Primes")

else:

print("Not Twin Primes")
```

```python
import math
def is_prime(n):
    """
    Check whether a number is prime using optimized logic.
    Parameters:
    n (int):# The number to check.
    Returns:
    bool: True if n is prime, otherwise False.
    """
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0:
        return False
    # Check only up to sqrt(n)
    for i in range(3, int(math.sqrt(n)) + 1, 2):
        if n % i == 0:
            return False
    return True
def is_twin_prime(p1, p2):
    """
    Check whether two numbers form a twin prime pair.
    Parameters:
    p1 (int): First number.
    p2 (int): Second number.
    Returns:
    bool: True if both are prime and differ by 2, otherwise False.
    """
    return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
def generate_twin_primes(start, end):
    """
    Generate all twin prime pairs within a given range.
    Parameters:
    start (int): Starting number of the range.
    end (int): Ending number of the range.
    Returns:
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Check specific pair (11, 13):
Twin Primes

Twin primes between 1 and 50:
[(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43)]
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 10 – Refactoring the Chinese Zodiac Program

Objective

Refactor the given poorly structured Python script into a clean, modular, and reusable implementation.

The current program reads a year from the user and prints the corresponding Chinese Zodiac sign. However, the implementation contains repetitive

conditional logic, lacks modular design, and does not follow clean coding principles.

Your task is to refactor the code to improve readability, maintainability, and structure.

Chinese Zodiac Cycle (Repeats Every 12 Years)

1. Rat

2. Ox

3. Tiger

4. Rabbit

5. Dragon

6. Snake

7. Horse

8. Goat (Sheep)

9. Monkey

10. Rooster

11. Dog

12. Pig

```
# Chinese Zodiac Program (Unstructured Version)

# This code needs refactoring.

year = int(input("Enter a year: "))

if year % 12 == 0:

print("Monkey")
```

```
elif year % 12 == 1:
print("Rooster")
elif year % 12 == 2:
print("Dog")
elif year % 12 == 3:
print("Pig")
elif year % 12 == 4:
print("Rat")
elif year % 12 == 5:
print("Ox")
elif year % 12 == 6:
print("Tiger")
elif year % 12 == 7:
print("Rabbit")
elif year % 12 == 8:
print("Dragon")
elif year % 12 == 9:
print("Snake")
elif year % 12 == 10:
print("Horse")
elif year % 12 == 11:
print("Goat")
```

You must:

1. Create a reusable function: get_zodiac(year)

2. Replace the if-elif chain with a cleaner structure (e.g., list or dictionary).

3. Add proper docstrings.

4. Separate input handling from logic.

5. Improve readability and maintainability.

6. Ensure output remains correct.

ass_13.py > ...

```python
#task-10
def get_zodiac(year):
    """
    Determine the Chinese Zodiac sign for a given year.
    Parameters:
    year (int): The year for which the zodiac sign is required.
    Returns:
    str: The corresponding Chinese Zodiac sign.
    """
    zodiac_cycle = [
        "Monkey",   # 0
        "Rooster",  # 1
        "Dog",      # 2
        "Pig",      # 3
        "Rat",      # 4
        "Ox",       # 5
        "Tiger",    # 6
        "Rabbit",   # 7
        "Dragon",   # 8
        "Snake",    # 9
        "Horse",    # 10
        "Goat"      # 11
    ]
    return zodiac_cycle[year % 12]
def main():
    """
    Handle user input and display the zodiac sign.
    """
    year = int(input("Enter a year: "))
    zodiac_sign = get_zodiac(year)
    print(zodiac_sign)
if __name__ == "__main__":
    main()

```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Enter a year: 2004
Monkey
```

Task 11 – Refactoring the Harshad (Niven) Number Checker

Refactor the given poorly structured Python script into a clean, modular, and reusable implementation.

A Harshad (Niven) number is a number that is divisible by the sum of its digits.

For example:

• 18 → 1 + 8 = 9 → 18 ÷ 9 = 2 (Harshad Number)

• 19 → 1 + 9 = 10 → 19 ÷ 10 ≠ integer (Not Harshad)

Problem Statement

The current implementation:

• Mixes logic and input handling

• Uses redundant variables

• Does not use reusable functions properly

• Returns print statements instead of boolean values

• Lacks documentation

You must refactor the code to follow clean coding principles.

```
# Harshad Number Checker (Unstructured Version)

num = int(input("Enter a number: "))

temp = num

sum_digits = 0
```

```
while temp > 0:

digit = temp % 10

sum_digits = sum_digits + digit

temp = temp // 10

if sum_digits != 0:

if num % sum_digits == 0:

print("True")

else:

print("False")

else:

print("False")
```

You must:

1. Create a reusable function: is_harshad(number)

2. The function must:

o Accept an integer parameter.

o Return True if the number is divisible by the sum of its digits.

o Return False otherwise.

3. Separate user input from core logic.

4. Add proper docstrings.

5. Improve readability and maintainability.

6. Ensure the program handles edge cases (e.g., 0, negative numbers).

ass_13.py > ...

```python
#task-11
def is_harshad(number):
    """
    Determine whether a number is a Harshad (Niven) number.
    A Harshad number is divisible by the sum of its digits.
    Parameters:
    number (int): The integer to check.
    Returns:
    bool: True if the number is a Harshad number,
          False otherwise.
    Edge Cases:
    - Returns False for 0 (division undefined).
    - Works with negative numbers by using absolute value.
    """
    if number == 0:
        return False
    num = abs(number)
    digit_sum = sum(int(digit) for digit in str(num))
    if digit_sum == 0:
        return False
    return num % digit_sum == 0
def main():
    """
    Handle user input and display whether the number
    is a Harshad number.
    """
    number = int(input("Enter a number: "))
    result = is_harshad(number)
    print(result)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Enter a number: 15
False
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 12 – Refactoring the Factorial Trailing Zeros Program

Refactor the given poorly structured Python script into a clean, modular, and efficient implementation.

The program calculates the number of trailing zeros in n! (factorial of n).

Problem Statement

The current implementation:

• Calculates the full factorial (inefficient for large n)

• Mixes input handling with business logic

• Uses print statements instead of return values

• Lacks modular structure and documentation

You must refactor the code to improve efficiency, readability, and maintainability.

```
# Factorial Trailing Zeros (Unstructured Version)
n = int(input("Enter a number: "))
fact = 1
i = 1
while i <= n:
fact = fact * i
i = i + 1
count = 0
while fact % 10 == 0:
count = count + 1
fact = fact // 10
print("Trailing zeros:", count)
```

You must:

1. Create a reusable function: count_trailing_zeros(n)

2. The function must:

o Accept a non-negative integer n.

o Return the number of trailing zeros in n!.

3. Do NOT compute the full factorial.

4. Use an optimized mathematical approach (count multiples of 5).

5. Add proper docstrings.

6. Separate user interaction from core logic.

7. Handle edge cases (e.g., negative numbers, zero).

Test Cases Design

ass_13.py > ...

```python
#task-12
def count_trailing_zeros(n):
    """
    Calculate the number of trailing zeros in n! (factorial of n).
    The function uses an optimized mathematical approach by
    counting multiples of 5 instead of computing the full factorial.
    Parameters:
    n (int): A non-negative integer.
    Returns:
    int: The number of trailing zeros in n!.
    Raises:
    ValueError: If n is negative.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    count = 0
    divisor = 5
    while n // divisor > 0:
        count += n // divisor
        divisor *= 5
    return count
def main():
    """
    Handle user input and display trailing zeros result.
    """
    n = int(input("Enter a number: "))
    try:
        result = count_trailing_zeros(n)
        print("Trailing zeros:", result)
    except ValueError as e:
        print(e)
if __name__ == "__main__":
    main()
```

```
PS C:\Users\2303a\OneDrive\Desktop\AI> & C:\Users\2303a\miniconda3\python.exe c:/Users/2303a/OneDrive/Desktop/AI/ass_13.py
Enter a number: 12
Trailing zeros: 2
PS C:\Users\2303a\OneDrive\Desktop\AI>
```

Task 13 (Collatz Sequence Generator – Test Case Design)

• Function: Generate Collatz sequence until reaching 1.

• Test Cases to Design:

• Normal: 6 → [6,3,10,5,16,8,4,2,1]

• Edge: 1 → [1]

• Negative: -5

• Large: 27 (well-known long sequence)

• Requirement: Validate correctness with pytest.

Explanation:

We need to write a function that:

• Takes an integer n as input.

• Generates the Collatz sequence (also called the 3n+1 sequence).

• The rules are:

o If n is even → next = n / 2.

o If n is odd → next = 3n + 1.

• Repeat until we reach 1.

• Return the full sequence as a list.

Example

Input: 6

Steps:

• 6 (even → 6/2 = 3)

• 3 (odd → 3*3+1 = 10)

• 10 (even → 10/2 = 5)

• 5 (odd → 3*5+1 = 16)

• 16 (even → 16/2 = 8)

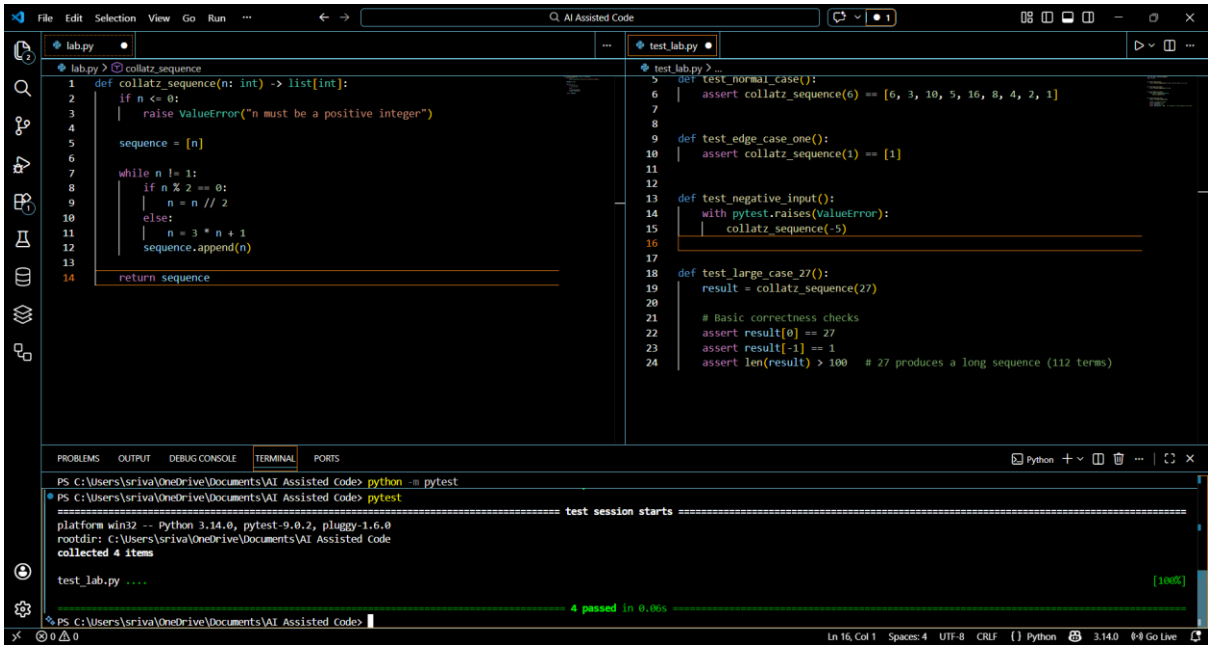
• 8 (even → 8/2 = 4)

• 4 (even → 4/2 = 2)

• 2 (even → 2/2 = 1)

Output:

[6, 3, 10, 5, 16, 8, 4, 2, 1]



Task 14 (Lucas Number Sequence – Test Case Design)

• Function: Generate Lucas sequence up to n terms.

(Starts with 2,1, then Fn = Fn-1 + Fn-2)

• Test Cases to Design:

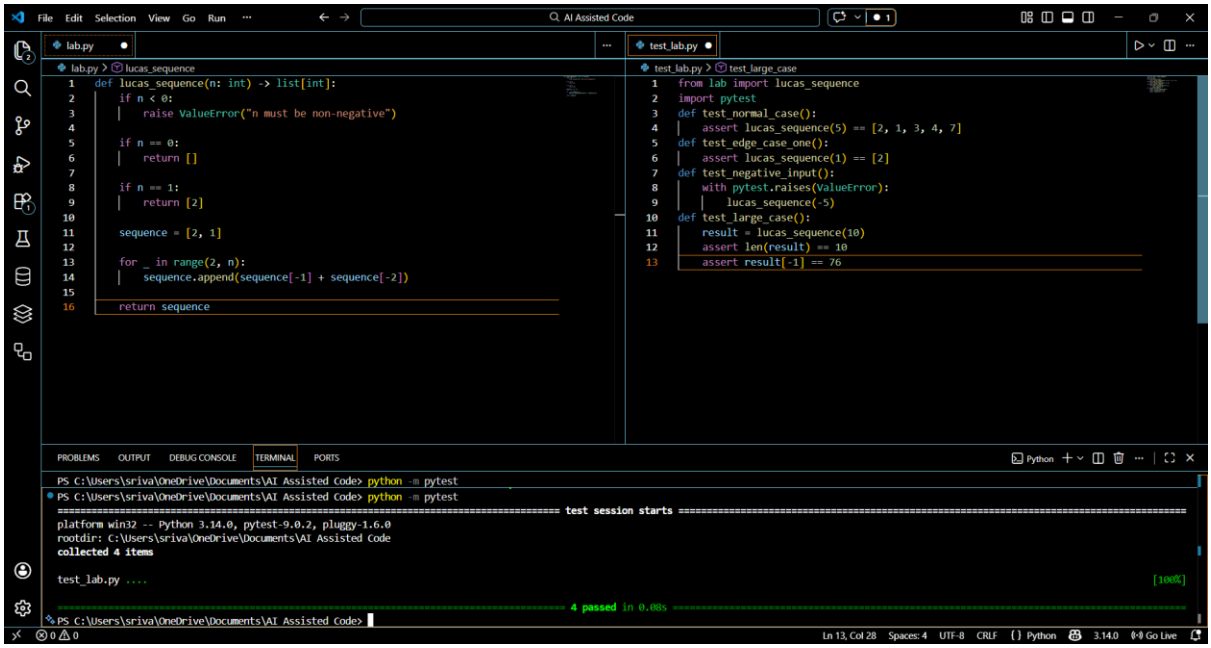
• Normal: 5 → [2, 1, 3, 4, 7]

• Edge: 1 → [2]

• Negative: -5 → Error

• Large: 10 (last element = 76).

• Requirement: Validate correctness with pytest.



Task 15 (Vowel & Consonant Counter – Test Case Design)

• Function: Count vowels and consonants in string.

• Test Cases to Design:

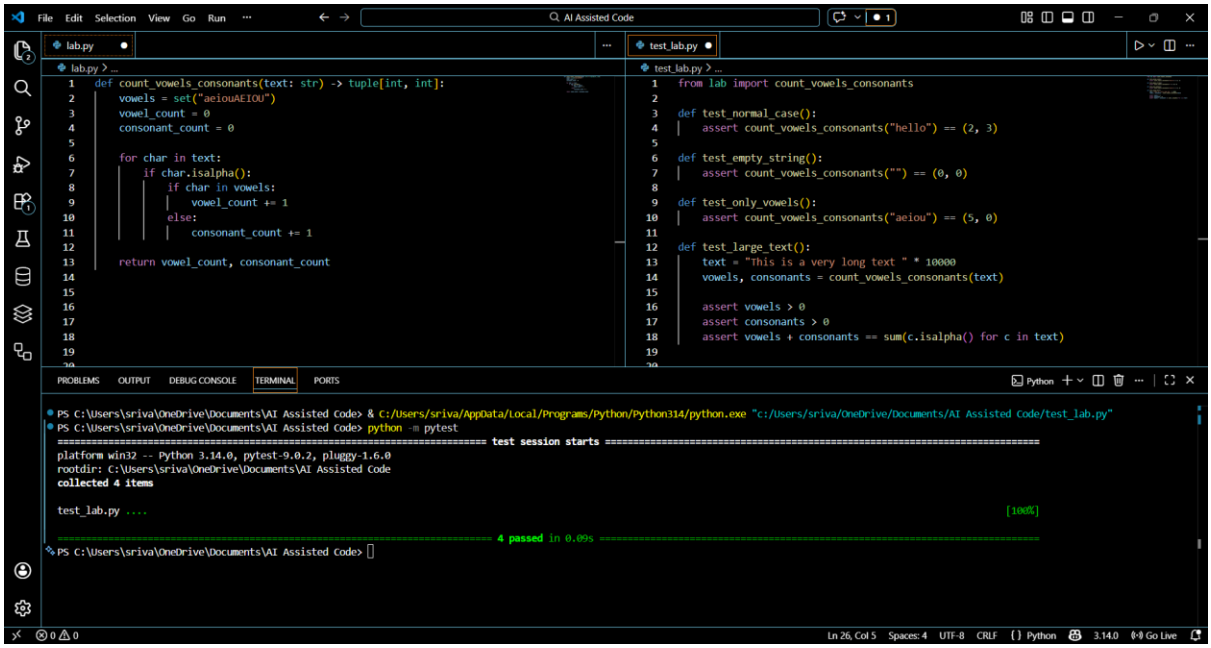
• Normal: "hello" → (2,3)

• Edge: "" → (0,0)

• Only vowels: "aeiou" → (5,0)

Large: Long text

• Requirement: Validate correctness with pytest.


File Edit Selection View Go Run ···
AI Assisted Code
lab.py
test_lab.py
lab.py > ...
test_lab.py > ...
def count_vowels_consonants(text: str) -> tuple[int, int]:
vowels = set("aeiouAEIOU")
vowel_count = 0
consonant_count = 0
for char in text:
if char.isalpha():
if char in vowels:
vowel_count += 1
else:
consonant_count += 1
return vowel_count, consonant_count
from lab import count_vowels_consonants
def test_normal_case():
assert count_vowels_consonants("hello") == (2, 3)
def test_empty_string():
assert count_vowels_consonants("") == (0, 0)
def test_only_vowels():
assert count_vowels_consonants("aeiou") == (5, 0)
def test_large_text():
text = "This is a very long text " * 10000
vowels, consonants = count_vowels_consonants(text)
assert vowels > 0
assert consonants > 0
assert vowels + consonants == sum(c.isalpha() for c in text)
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> & C:/Users/sriva/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/sriva/OneDrive/Documents/AI Assisted Code/test_lab.py"
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code> python -m pytest
test session starts
platform win32 -- Python 3.14.0, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\sriva\OneDrive\Documents\AI Assisted Code
collected 4 items
test_lab.py ....
[100%]
4 passed in 0.09s
PS C:\Users\sriva\OneDrive\Documents\AI Assisted Code>
Ln 26, Col 5 Spaces: 4 UTF-8 CRLF Python 3.14.0 Go Live