AI ASSISTED CODING

ASSIGNMENT-13

Name: Harshitha Guda

H.T.No:2303A51102

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.

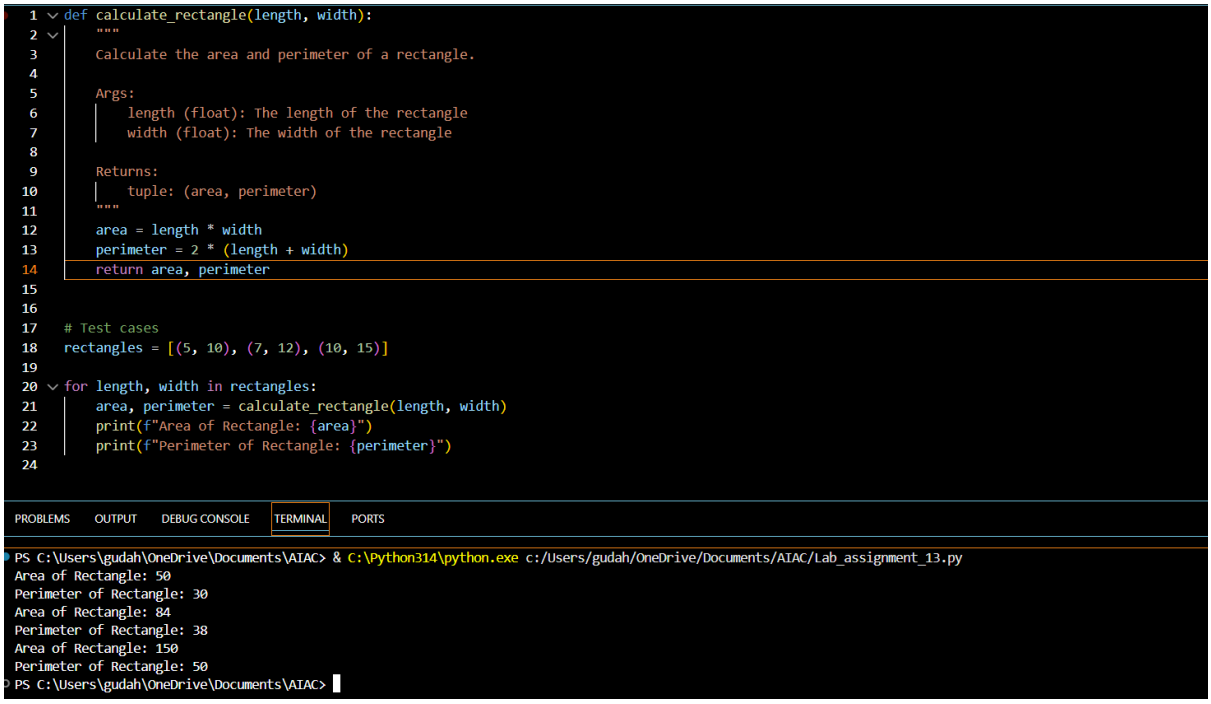
• Sample Legacy Code:

```
# 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



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:

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

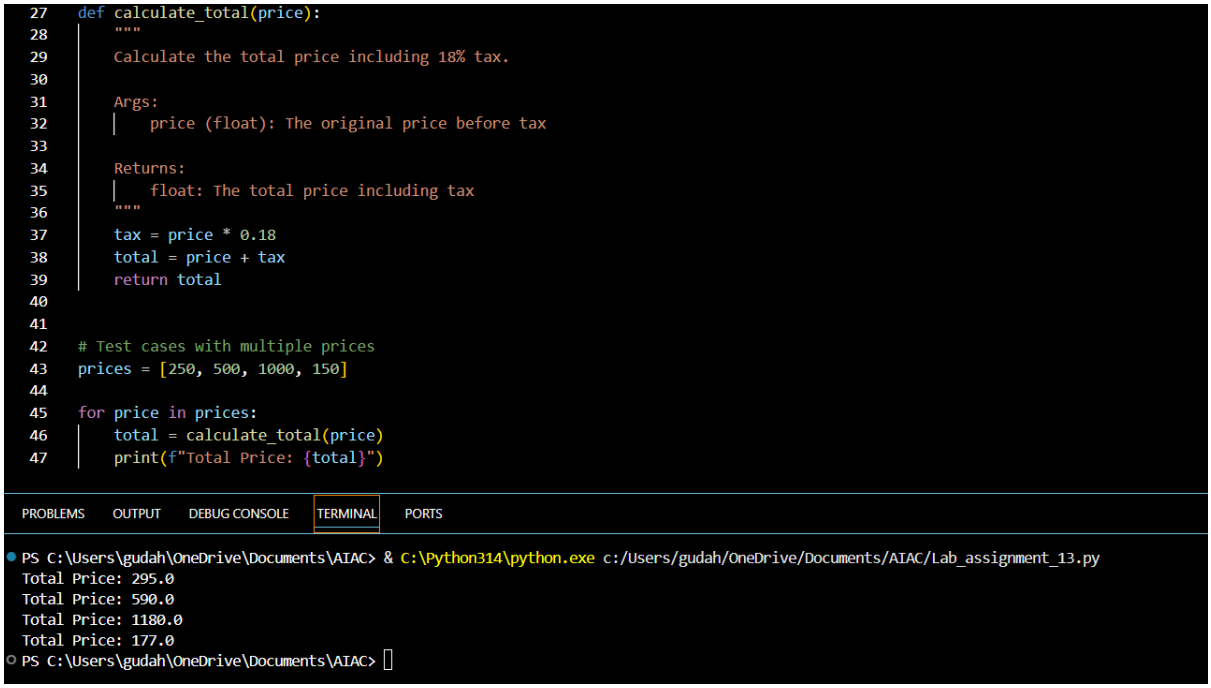
total = price + tax

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



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.

```python
class GradeCalculator:
    def calculate_grade(self, marks):
        if marks >= 90:
            return "Grade A"
        elif marks >= 75:
            return "Grade B"
        else:
            return "Grade C"

calculator = GradeCalculator()
student_marks = [85, 92, 70, 88, 95]

for marks in student_marks:
    grade = calculator.calculate_grade(marks)
    print(f"Marks: {marks}, {grade}")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Marks: 85, Grade B
Marks: 92, Grade A
Marks: 70, Grade C
Marks: 88, Grade B
Marks: 95, Grade A
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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: "))

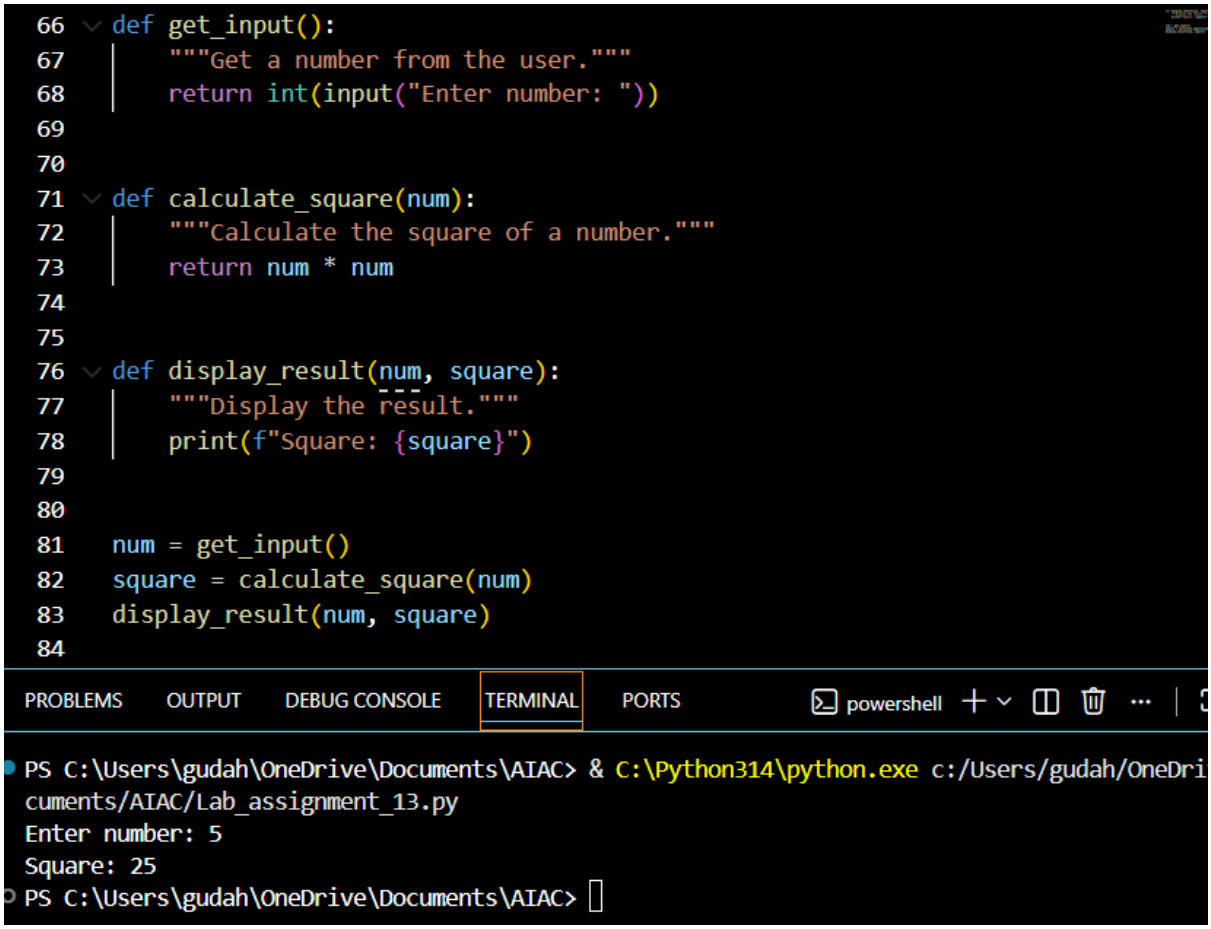
square = num * num

print("Square:", square)

• Expected Output:

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

display_result().



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.

```python
class EmployeeSalaryCalculator:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        self.tax_rate = 0.2

    def calculate_net_salary(self):
        tax = self.salary * self.tax_rate
        return self.salary - tax

    def calculate_bonus(self, bonus_percent):
        return self.salary * (bonus_percent / 100)

    def display_salary_info(self):
        net = self.calculate_net_salary()
        print(f"Employee: {self.name}")
        print(f"Gross Salary: {self.salary}")
        print(f"Net Salary: {net}")

# Test
salary = 50000
employee = EmployeeSalaryCalculator("John", salary)
employee.display_salary_info()
bonus = employee.calculate_bonus(10)
print(f"Bonus (10%): {bonus}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Employee: John
Gross Salary: 50000
Net Salary: 40000.0
Bonus (10%): 5000.0
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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:

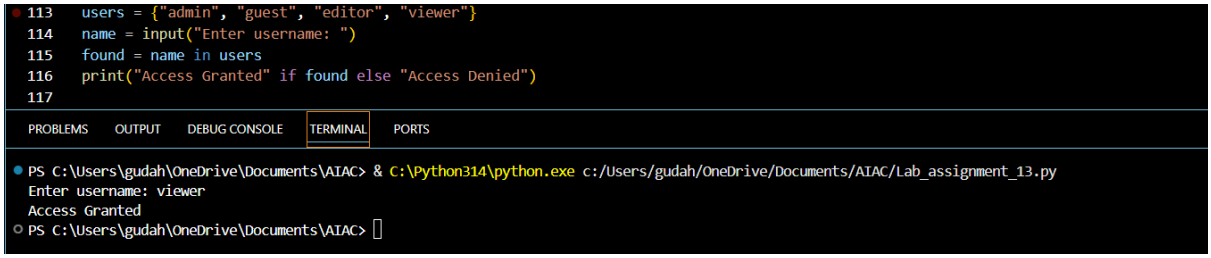
if u == name:

found = True

print("Access Granted" if found else "Access Denied")

Expected Outcome:

o Use of sets or dictionaries with complexity justification



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.")

```
"""
A simple library management system that allows adding, removing, and searching for books.
Each book is identified by its ISBN, and the library database is stored in a dictionary.
The functions provided include:
- add_book: Adds a book to the library database.
- remove_book: Removes a book from the library database by ISBN.
- search_book: Searches for a book in the library database by ISBN and returns its details if found.
The code also includes test cases to demonstrate the functionality of the library management system."""
library_db = {}
def add_book(title, author, isbn):
    """Add 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 identifier for the book
    """
    if isbn not in library_db:
        library_db[isbn] = {"title": title, "author": author}
        print("Book added successfully.")
    else:
        print("Book already exists.")

def remove_book(isbn):
    """Remove a book from the library database by ISBN.

    Args:
        isbn (str): The ISBN identifier of the book to remove
    """
    if isbn in library_db:
        del library_db[isbn]
        print("Book removed successfully.")
    else:
        print("Book not found.")

```

```
def search_book(isbn):
    """Search for a book in the library database by ISBN.

    Args:
        isbn (str): The ISBN identifier of the book to search

    Returns:
        dict: Book details if found, None otherwise
    """
    if isbn in library_db:
        print("Book Found:", library_db[isbn])
        return library_db[isbn]
    else:
        print("Book not found.")
        return None

# Test the functions
add_book("Python Basics", "John Doe", "101")
add_book("AI Fundamentals", "Jane Smith", "102")
search_book("101")
remove_book("101")
search_book("101")
print("Book already exists.")

```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/library.py
Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
Book already exists.
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pydoc library
Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
Book already exists.
Help on module library:

NAME
    library

DESCRIPTION
    A simple library management system that allows adding, removing, and searching for books.
    Each book is identified by its ISBN, and the library database is stored in a dictionary.
    The functions provided include:
    - add_book: Adds a book to the library database.
    - remove_book: Removes a book from the library database by ISBN.
    - search_book: Searches for a book in the library database by ISBN and returns its details if found.
    The code also includes test cases to demonstrate the functionality of the library management system.

FUNCTIONS
    add_book(title, author, isbn)
        Add 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 identifier for the book

-- More --
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> python -m pydoc -w  library
Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
Book already exists.
wrote library.html
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

index

library c:\users\gudah\onedrive\documents\aiac\library.py

A simple library management system that allows adding, removing, and searching for books.
Each book is identified by its ISBN, and the library database is stored in a dictionary.
The functions provided include:
- add_book: Adds a book to the library database.
- remove_book: Removes a book from the library database by ISBN.
- search_book: Searches for a book in the library database by ISBN and returns its details if found.
The code also includes test cases to demonstrate the functionality of the library management system.

Functions

add_book(title, author, isbn)
Add 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 identifier for the book

remove_book(isbn)
Remove a book from the library database by ISBN.

Args:
isbn (str): The ISBN identifier of the book to remove

search_book(isbn)
Search for a book in the library database by ISBN.

Args:
isbn (str): The ISBN identifier of the book to search

Returns:
dict: Book details if found, None otherwise

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
```

```python
def generate_fibonacci(limit):
    """Generate Fibonacci sequence up to a given limit.
    Args:
        limit (int): The number of Fibonacci numbers to generate
    Returns:
        list: A list of Fibonacci numbers"""
    if limit <= 0:
        return []
    elif limit == 1:
        return [0]
    fib_sequence = [0, 1]
    for i in range(2, limit):
        next_fib = fib_sequence[i-1] + fib_sequence[i-2]
        fib_sequence.append(next_fib)
    return fib_sequence
# Test cases
if __name__ == "__main__":
    # Test 1: Generate first 10 Fibonacci numbers
    result = generate_fibonacci(10)
    print(f"Fibonacci (limit=10): {result}")
    # Test 2: Edge case - limit of 0
    result = generate_fibonacci(0)
    print(f"Fibonacci (limit=0): {result}")
    # Test 3: Edge case - limit of 1
    result = generate_fibonacci(1)
    print(f"Fibonacci (limit=1): {result}")
    # Test 4: User input
    n = int(input("Enter limit: "))
    fib_numbers = generate_fibonacci(n)
    for num in fib_numbers:
        print(num)

```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Fibonacci (limit=10): [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
Fibonacci (limit=0): []
Fibonacci (limit=1): [0]
Enter limit: 8
0
1
1
2
3
5
8
13
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

Comparison:

1. **Approach**: The commented function is reusable and returns a complete list; the active script prints values iteratively using a simple loop.
2. **Input Validation**: The function includes edge cases (limit ≤ 0, limit = 1); the script doesn't validate input.
3. **Storage**: The function stores all Fibonacci numbers in a list; the script only keeps the last two values in memory.
4. **Reusability**: The function can be called multiple times for different limits; the script requires re-running for each calculation.
5. **Output**: The function returns a list for further processing; the script directly prints, making it harder to reuse the results.

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
def is_prime(n):
    """
    Check if a number is prime.
    Args:
        n (int): The number to check
    Returns:
        bool: True if prime, False otherwise
    """
    if n < 2:
        return False
    if n == 2:
        return True
    if n % 2 == 0:
        return False
    for i in range(3, int(n**0.5) + 1, 2):
        if n % i == 0:
            return False
    return True
def is_twin_prime(p1, p2):
    """
    Check if two numbers are twin primes.
    Args:
        p1 (int): First number
        p2 (int): Second number
    Returns:
        bool: True if both are prime and differ by 2
    """
    return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2

```

```python
def find_twin_primes(start, end):
    """
    Generate a list of twin prime pairs in a given range.
    Args:
        start (int): Start of range
        end (int): End of range
    Returns:
        list: List of twin prime tuples
    """
    twins = []
    for num in range(start, end - 1):
        if is_twin_prime(num, num + 2):
            twins.append((num, num + 2))
    return twins
# Test cases
if __name__ == "__main__":
    a, b = 11, 13
    if is_twin_prime(a, b):
        print("Twin Primes")
    else:
        print("Not Twin Primes")

    # Find twin primes in range
    twin_primes = find_twin_primes(1, 50)
    print(f"Twin primes in range (1-50): {twin_primes}")
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Twin Primes
Twin primes in range (1-50): [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43)]
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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)

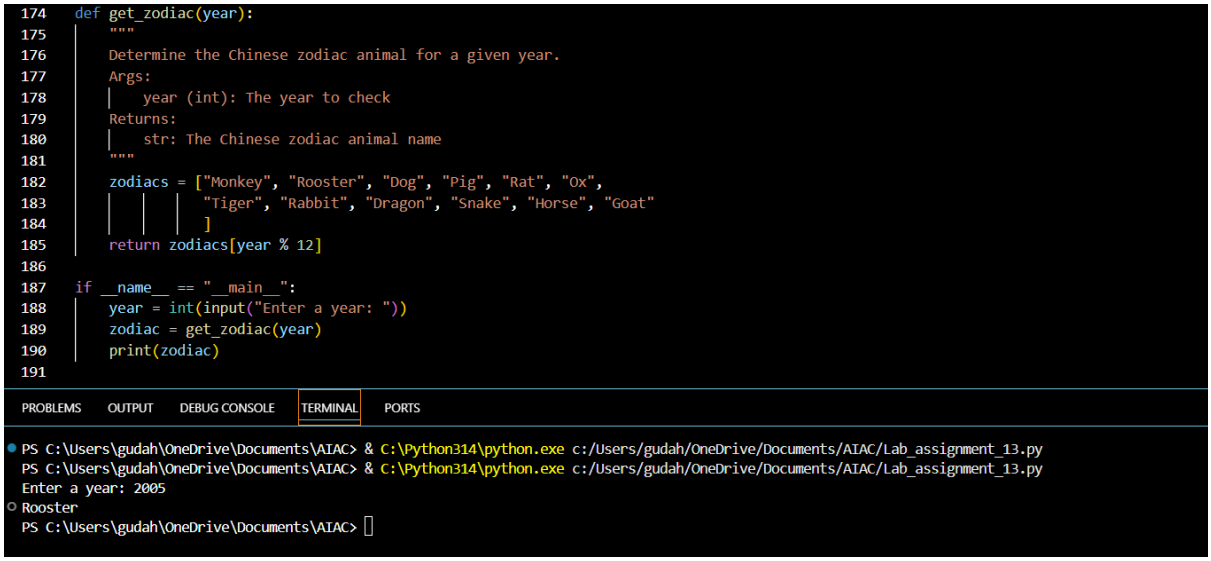
2. Replace the if-elif chain with a cleaner structure (e.g., list or dictionary).

3. Add proper docstrings.

4. Separate input handling from logic.

5. Improve readability and maintainability.

6. Ensure output remains correct.



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).

```
def is_harshad(number):
    """
    Check if a number is a Harshad number.
    A Harshad number is divisible by the sum of its digits.
    Args:
        number (int): The number to check
    Returns:
        bool: True if the number is a Harshad number, False otherwise
    """
    if number <= 0:
        return False
    digit_sum = sum(int(digit) for digit in str(number))
    return number % digit_sum == 0
if __name__ == "__main__":
    num = int(input("Enter a number: "))
    if is_harshad(num):
        print("True")
    else:
        print("False")
```

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Enter a number: 456
False
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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!.

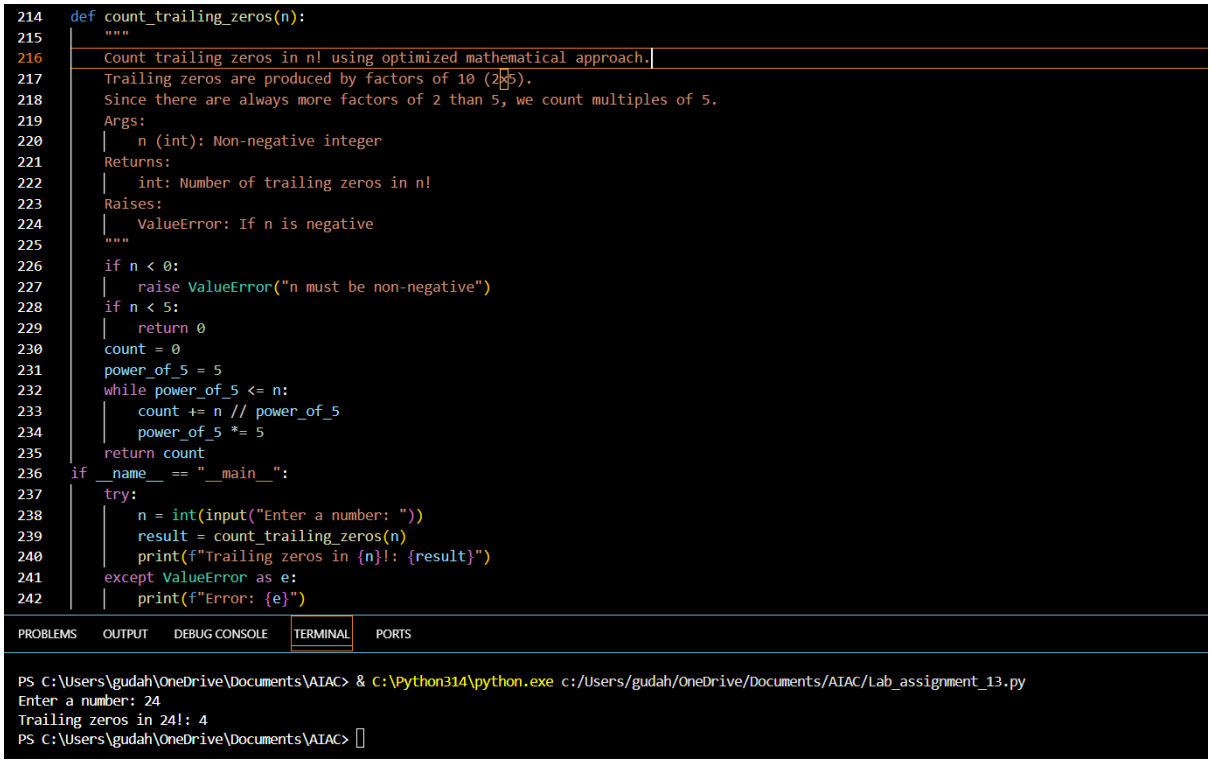
3. Do NOT compute the full factorial.

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).



```python
def count_trailing_zeros(n):
    """
    Count trailing zeros in n! using optimized mathematical approach.
    Trailing zeros are produced by factors of 10 (2×5).
    Since there are always more factors of 2 than 5, we count multiples of 5.
    Args:
        n (int): Non-negative integer
    Returns:
        int: Number of trailing zeros in n!
    Raises:
        ValueError: If n is negative
    """
    if n < 0:
        raise ValueError("n must be non-negative")
    if n < 5:
        return 0
    count = 0
    power_of_5 = 5
    while power_of_5 <= n:
        count += n // power_of_5
        power_of_5 *= 5
    return count
if __name__ == "__main__":
    try:
        n = int(input("Enter a number: "))
        result = count_trailing_zeros(n)
        print(f"Trailing zeros in {n}!: {result}")
    except ValueError as e:
        print(f"Error: {e}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
Enter a number: 24
Trailing zeros in 24!: 4
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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]

```python
def collatz_sequence(n):
    """
    Generate the Collatz sequence starting from n.
    Args:
        n (int): The starting number
    Returns:
        list: The Collatz sequence from n to 1
    """
    sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        sequence.append(n)
    return sequence
def test_collatz_sequence_small_even():
    """Test collatz_sequence with a small even number."""
    assert collatz_sequence(6) == [6, 3, 10, 5, 16, 8, 4, 2, 1]
def test_collatz_sequence_small_odd():
    """Test collatz_sequence with a small odd number."""
    assert collatz_sequence(5) == [5, 16, 8, 4, 2, 1]
def test_collatz_sequence_one():
    """Test collatz_sequence with 1 (already at end)."""
    assert collatz_sequence(1) == [1]
def test_collatz_sequence_power_of_two():
    """Test collatz_sequence with a power of two."""
    assert collatz_sequence(8) == [8, 4, 2, 1]
def test_collatz_sequence_larger_number():
    """Test collatz_sequence with a larger number."""
    result = collatz_sequence(10)
    assert result[0] == 10
    assert result[-1] == 1
    assert len(result) > 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.

```python
def lucas_sequence(n):
    """
    Generate Lucas sequence up to n terms.
    Args:n (int): Number of terms to generate
    Returns:list: Lucas sequence up to n terms
    Raises:ValueError: If n is negative
    """
    if n < 0:
        raise ValueError("n must be non-negative")
    if n == 0:
        return []
    if n == 1:
        return [2]
    sequence = [2, 1]
    for _ in range(n - 2):
        sequence.append(sequence[-1] + sequence[-2])
    return sequence
# Test cases
if __name__ == "__main__":
    def test_normal_case():
        assert lucas_sequence(5) == [2, 1, 3, 4, 7]
    def test_single_term():
        assert lucas_sequence(1) == [2]
    def test_negative():
        with pytest.raises(ValueError):
            lucas_sequence(-5)
    def test_large_case():
        result = lucas_sequence(10)
        assert result[-1] == 76
    def test_empty():
        assert lucas_sequence(0) == []
    test_normal_case()
    test_single_term()
    test_negative()
    test_large_case()
    test_empty()
    print("All tests passed!")
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/Lab_assignment_13.py
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/library.py
All tests passed!
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```

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.

```python
def count_vowels_and_consonants(text):
    """
    Count vowels and consonants in a string.
    Args:text (str): The input string to analyze
    Returns:tuple: (vowel_count, consonant_count)
    """
    vowels = "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)
def test_normal_case():
    assert count_vowels_and_consonants("hello") == (2, 3)
def test_empty_string():
    assert count_vowels_and_consonants("") == (0, 0)
def test_only_vowels():
    assert count_vowels_and_consonants("aeiou") == (5, 0)
def test_large_text():
    long_text = "The quick brown fox jumps over the lazy dog"
    result = count_vowels_and_consonants(long_text)
    assert result == (11, 24)
def test_mixed_case():
    assert count_vowels_and_consonants("AEIOUaeiou") == (10, 0)
if __name__ == "__main__":
    test_normal_case()
    test_empty_string()
    test_only_vowels()
    test_large_text()
    test_mixed_case()
    print("All tests passed!")
```

```
PS C:\Users\gudah\OneDrive\Documents\AIAC> & C:\Python314\python.exe c:/Users/gudah/OneDrive/Documents/AIAC/library.py
All tests passed!
PS C:\Users\gudah\OneDrive\Documents\AIAC>
```