

Assignment-13

Ht. No:2303A51923

Name: V.Sravani

Batch: 23

Friday

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

```

Ass13.py > ...
1  """print("Area of Rectangle:", 5 * 10)
2  print("Perimeter of Rectangle:", 2 * (5 + 10))
3  print("Area of Rectangle:", 7 * 12)
4  print("Perimeter of Rectangle:", 2 * (7 + 12))
5  print("Area of Rectangle:", 10 * 15)
6  print("Perimeter of Rectangle:", 2 * (10 + 15))"""
7  #refactor the above code with a reusable function and handle the duplicates
8  def calculate_rectangle_properties(length, width):
9      """Calculate the area and perimeter of a rectangle.
10     parameters:
11         length (float): The length of the rectangle.
12         width (float): The width of the rectangle.
13     returns:
14         tuple: A tuple containing the area and perimeter of the rectangle.
15     handle the duplicates and make the code reusable.
16     """
17     area = length * width
18     perimeter = 2 * (length + width)
19     return area, perimeter
20 # Example usage:
21 rectangles = [(5, 10), (7, 12), (10, 15)]
22 for length, width in rectangles:
23     area, perimeter = calculate_rectangle_properties(length, width)
24     print(f"Area of Rectangle: {area}")
25     print(f"Perimeter of Rectangle: {perimeter}")
26

```

```

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> &
xe "c:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/As
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\SRAVANI\OneDrive\Dokumen\AI Assist Coding>

```

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

```
13 2.py > ...
1  def calculate_total(price):
5     price (int or float): The original price of the product.
6     Returns:
7     float: The total price after adding 18% tax.
8     """
9     tax = price * 0.18
10    total = price + tax
11    return total
12  def main():
13    """
14    Main function to calculate and print total prices
15    for multiple product values.
16    Returns:
17    None
18    """
19    prices = [250, 500]
20    for price in prices:
21        total = calculate_total(price)
22        print("Total Price:", total)
23  if __name__ == "__main__":
24      main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/
Microsoft/OneDrive/OneDrive.exe "c:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/13 2.py"
Total Price: 295.0
Total Price: 590.0
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>
```

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 $\leq 100 \rightarrow$ "Grade A"
 - Marks $\geq 80 \rightarrow$ "Grade B"
 - Marks $\geq 70 \rightarrow$ "Grade C"
 - Marks $\geq 40 \rightarrow$ "Grade D"
 - Marks $\geq 0 \rightarrow$ "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.

```
gradecalculator.py > ...
1 class GradeCalculator:
2     """
3     A class used to calculate student grades based on marks.
4     Methods:
5     calculate_grade(self, marks):
6         Returns the grade string based on the given marks.
7     """
8     def calculate_grade(self, marks):
9         """
10        Determine the grade for a given mark.
11        Parameters:
12        marks (int or float): The student's marks (assumed between 0 and 100).
13        Returns:
14        str: The grade corresponding to the marks based on the grading rules.
15        """
16        if marks >= 90 and marks <= 100:
17            return "Grade A"
18        elif marks >= 80:
19            return "Grade B"
20        elif marks >= 70:
21            return "Grade C"
22        elif marks >= 40:
23            return "Grade D"
24        elif marks >= 0:
25            return "Fail"
26
27 def main():
28     """
29     Main function to demonstrate reusable grade calculation.
30     """
31     calculator = GradeCalculator()
32     marks_list = [85, 72, 95, 38]
33     for marks in marks_list:
34         grade = calculator.calculate_grade(marks)
35         print(f"Marks: {marks} -> {grade}")
36
37 if __name__ == "__main__":
38     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/User
xe "c:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/gradecal
Marks: 85 -> Grade B
Marks: 72 -> Grade C
Marks: 95 -> Grade A
Marks: 38 -> Fail
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>
```

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:

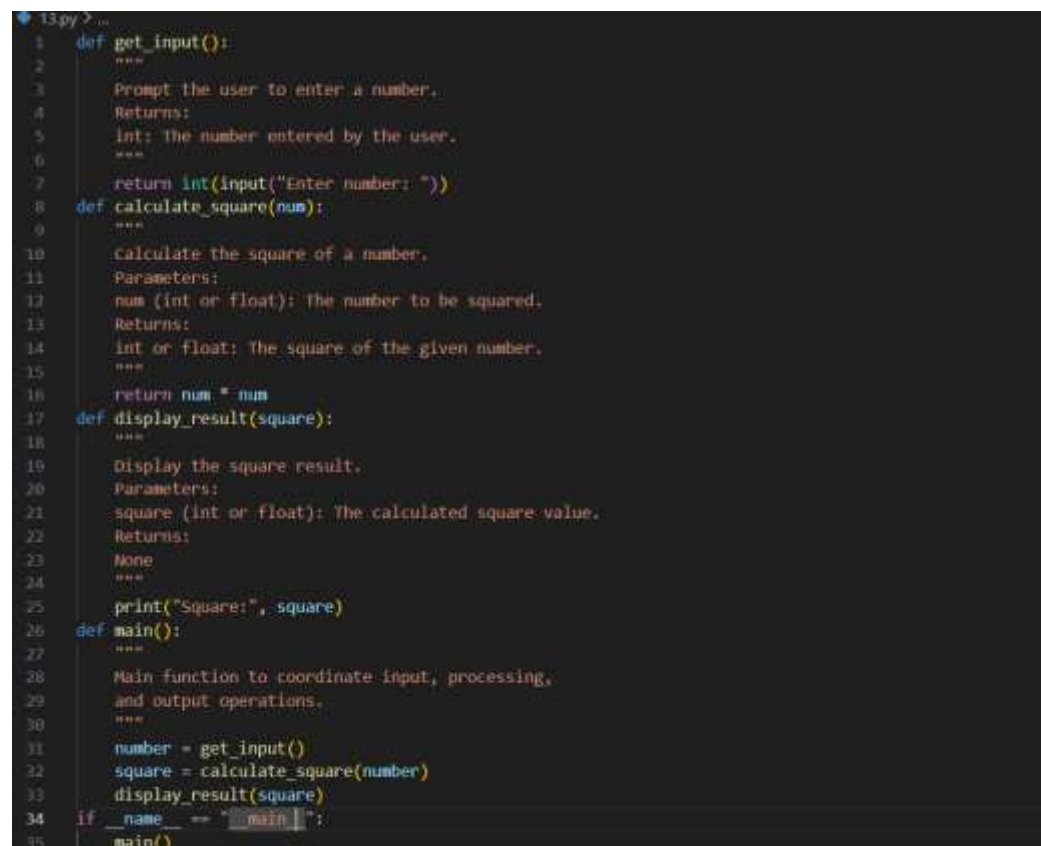
```
num = int(input("Enter number: "))
```

```
square = num * num
```

```
print("Square:", square)
```

- Expected Output:

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



```
13.py > ...
1  def get_input():
2      """
3      Prompt the user to enter a number.
4      Returns:
5      int: The number entered by the user.
6      """
7      return int(input("Enter number: "))
8  def calculate_square(num):
9      """
10     Calculate the square of a number.
11     Parameters:
12     num (int or float): The number to be squared.
13     Returns:
14     int or float: The square of the given number.
15     """
16     return num * num
17  def display_result(square):
18      """
19     Display the square result.
20     Parameters:
21     square (int or float): The calculated square value.
22     Returns:
23     None
24     """
25     print("Square:", square)
26  def main():
27      """
28     Main function to coordinate input, processing,
29     and output operations.
30     """
31     number = get_input()
32     square = calculate_square(number)
33     display_result(square)
34  if __name__ == "__main__":
35     main()
```

```

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData
AI Assist Coding/13.py"
Enter number: 20
Square: 400
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>

```

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.

```

1  class EmployeeSalaryCalculator:
17      def calculate_tax(self):
22          """
23          return self.salary * self.tax_rate
24      def calculate_net_salary(self):
25          """
26          Calculate the net salary after tax deduction.
27          Returns:
28          float: The net salary.
29          """
30          tax = self.calculate_tax()
31          return self.salary - tax
32      def display_net_salary(self):
33          """
34          Display the net salary.
35          Returns:
36          None
37          """
38          net_salary = self.calculate_net_salary()
39          print("Net Salary:", net_salary)
40      def main():
41          """
42          Main function to demonstrate salary calculation.
43          """
44          employee = EmployeeSalaryCalculator(50000)
45          employee.display_net_salary()
46      if __name__ == "__main__":
47          main()

```

```

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/13.py"
Net Salary: 40000.0
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>

```


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:
```

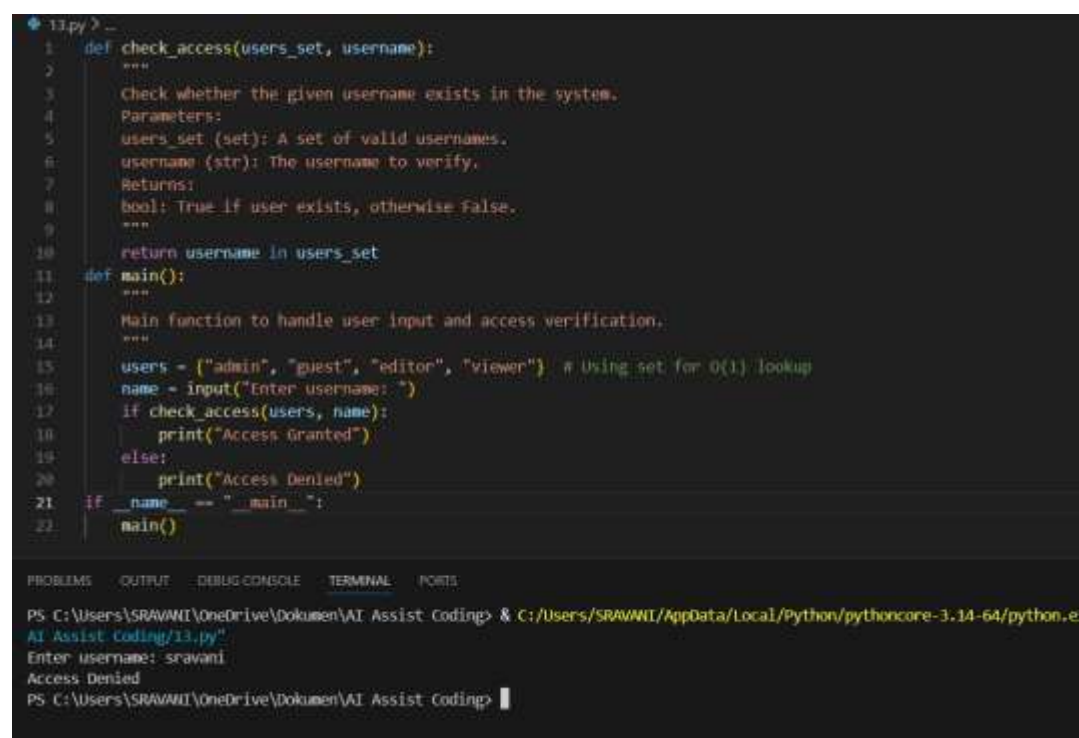
```
    if u == name:
```

```
        found = True
```

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

Expected Outcome:

- o Use of sets or dictionaries with complexity justification



```
13.py > _
1 def check_access(users_set, username):
2     """
3     Check whether the given username exists in the system.
4     Parameters:
5     users_set (set): A set of valid usernames.
6     username (str): The username to verify.
7     Returns:
8     bool: True if user exists, otherwise false.
9     """
10    return username in users_set
11
12 def main():
13    """
14    Main function to handle user input and access verification.
15    """
16    users = {"admin", "guest", "editor", "viewer"} # Using set for O(1) lookup
17    name = input("Enter username: ")
18    if check_access(users, name):
19        print("Access Granted")
20    else:
21        print("Access Denied")
22
23 if __name__ == "__main__":
24    main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\SRVMMI\OneDrive\Dokumen\AI Assist Coding> & C:\Users\SRVMMI\AppData\Local\Python\pythoncore-3.14-64\python.exe
AI Assist Coding/13.py
Enter username: sravani
Access Denied
PS C:\Users\SRVMMI\OneDrive\Dokumen\AI Assist Coding>
```

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.")
```

```

library.py > search_book
11 def add_book(title, author, isbn):
12     title (str): title of the book.
13     author (str): Author of the book.
14     isbn (str): Unique ISBN Identifier.
15
16     Returns:
17     str: Success or duplicate message.
18     """
19     if isbn not in library_db:
20         library_db[isbn] = {"title": title, "author": author}
21         return "Book added successfully."
22     return "Book already exists."
23
24 def remove_book(isbn):
25     """
26     Removes a book from the library database.
27
28     Parameters:
29     isbn (str): ISBN of the book to remove.
30
31     Returns:
32     str: Success or not found message.
33     """
34     if isbn in library_db:
35         del library_db[isbn]
36         return "Book removed successfully."
37     return "Book not found."
38
39
40
41
42

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python +

wrote 13.html

PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding> & C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64\python.exe "C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding\13.py"

PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding> python -m pydoc -p 1432

Server ready at http://localhost:1432/

Server commands: [b]rowser, [q]uit

server> http://localhost:1432/

C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding			
13.1	13.1-13.2	13.3	13.4
13.1	13.1-13.2	13.3	13.4
13.1	13.1-13.2	13.3	13.4
C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64\python314.zip			
C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64\DLLs			
13.1	13.1-13.2	13.3	13.4
13.1	13.1-13.2	13.3	13.4
13.1	13.1-13.2	13.3	13.4
13.1	13.1-13.2	13.3	13.4

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

```

1  def generate_fibonacci(n):
2      """
3      Generate Fibonacci series up to n terms.
4      Parameters:
5      n (int): Number of terms to generate. Must be >= 0.
6      Returns:
7      list: A list containing the Fibonacci sequence up to n terms.
8      """
9      if n <= 0:
10         return []
11     if n == 1:
12         return [0]
13     fibonacci_series = [0, 1]
14     for _ in range(2, n):
15         fibonacci_series.append(
16             fibonacci_series[-1] + fibonacci_series[-2]
17         )
18     return fibonacci_series
19
20 def main():
21     """
22     Main function to demonstrate Fibonacci generation.
23     """
24     n = int(input("Enter limit: "))
25     result = generate_fibonacci(n)
26     print("Fibonacci Series:", result)
27
28 if __name__ == "__main__":
29     main()

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe
AI Assist Coding/18-02-26.py
Enter limit: 20
Fibonacci Series: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584, 4181]
PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding>

```

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")
```

```
18-02-26.py > ...
2  def is_prime(n):
10     if n <= 1:
11         return False
12     if n <= 3:
13         return True
14     if n % 2 == 0:
15         return False
16     # Check only up to sqrt(n)
17     for i in range(3, int(math.sqrt(n)) + 1, 2):
18         if n % i == 0:
19             return False
20     return True
21  def is_twin_prime(p1, p2):
22      """
23      Check whether two numbers form a twin prime pair.
24      Parameters:
25      p1 (int): First number.
26      p2 (int): Second number.
27      Returns:
28      bool: True if both are prime and differ by 2, otherwise False.
29      """
30      return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
31  def generate_twin_primes(start, end):
32      """
33      Generate all twin prime pairs within a given range.
34      Parameters:
35      start (int): Starting number of the range.
36      end (int): Ending number of the range.
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

AI Assist Coding/18-02-26.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\SRVANT\OneDrive\Documents\AI Assist Coding>

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:
```

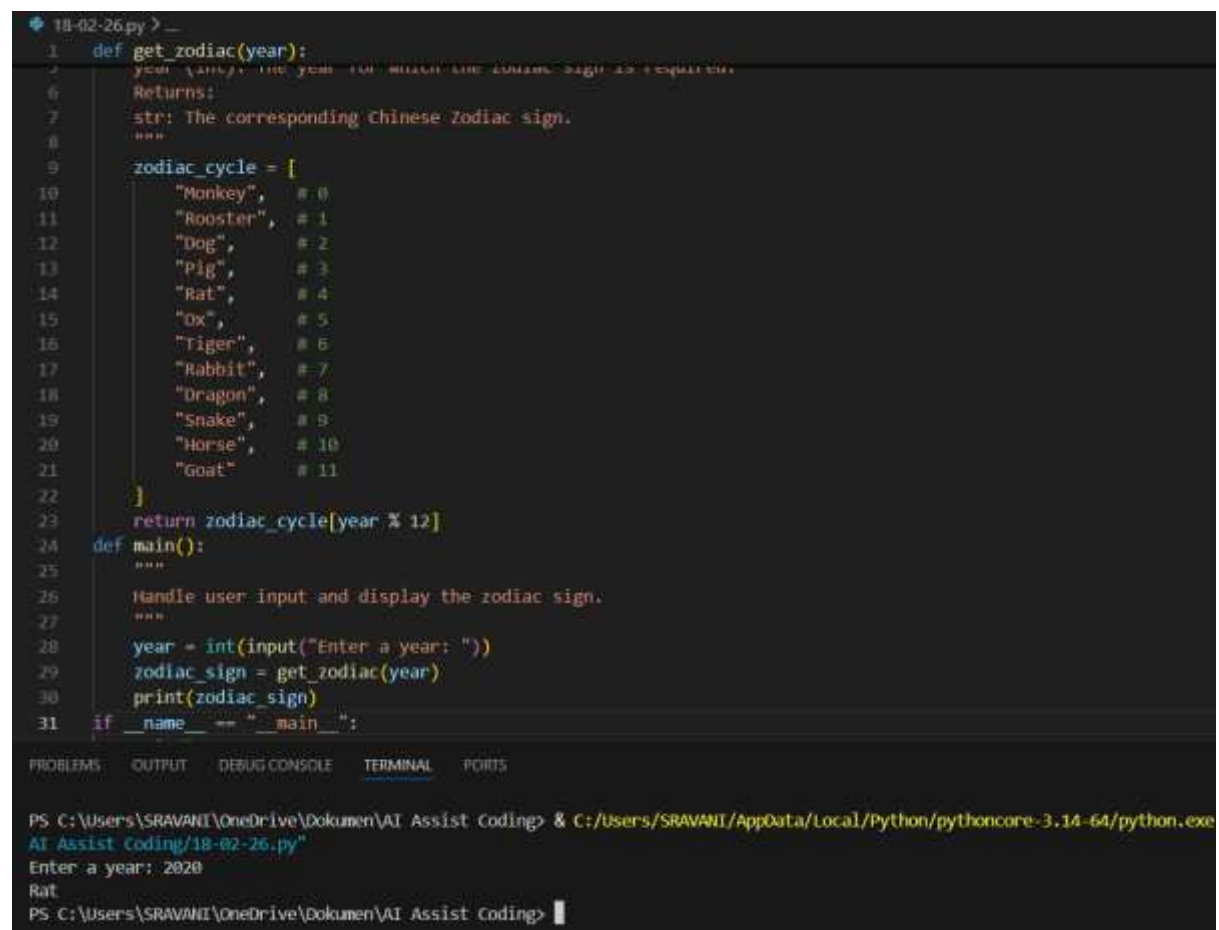
```
    print("Dragon")
```



```
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.



```
18-02-26.py > _
1 def get_zodiac(year):
2     """
3     year: int, the year for which the zodiac sign is required.
4     Returns:
5     str: The corresponding Chinese zodiac sign.
6     """
7     zodiac_cycle = [
8         "Monkey", # 0
9         "Rooster", # 1
10        "Dog", # 2
11        "Pig", # 3
12        "Rat", # 4
13        "Ox", # 5
14        "Tiger", # 6
15        "Rabbit", # 7
16        "Dragon", # 8
17        "Snake", # 9
18        "Horse", # 10
19        "Goat", # 11
20    ]
21    return zodiac_cycle[year % 12]
22
23 def main():
24     """
25     Handle user input and display the zodiac sign.
26     """
27     year = int(input("Enter a year: "))
28     zodiac_sign = get_zodiac(year)
29     print(zodiac_sign)
30
31 if __name__ == "__main__":
32     main()
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64\python.exe AI Assist Coding\18-02-26.py

Enter a year: 2020

Rat

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> |

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:

- $18 \rightarrow 1 + 8 = 9 \rightarrow 18 \div 9 = 2 \quad \square$ (Harshad Number)
- $19 \rightarrow 1 + 9 = 10 \rightarrow 19 \div 10 \neq \text{integer} \quad \square$ (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:
```

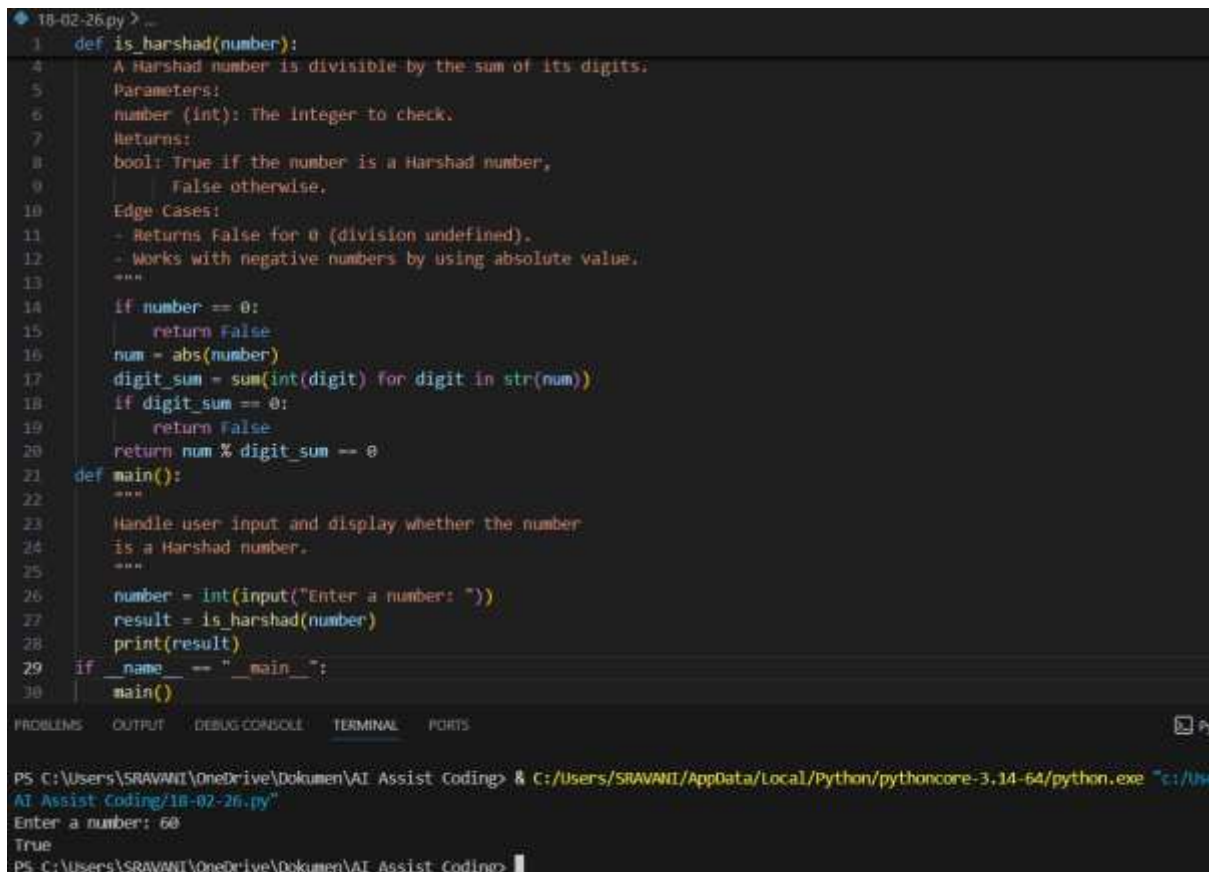
```
        print("False")
```

```
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).



```
18-02-26.py > ...
1  def is_harshad(number):
2      """
3      A Harshad number is divisible by the sum of its digits.
4      Parameters:
5      number (int): The integer to check.
6      Returns:
7      bool: True if the number is a Harshad number,
8           False otherwise.
9      Edge Cases:
10     - Returns False for 0 (division undefined).
11     - Works with negative numbers by using absolute value.
12     """
13
14     if number == 0:
15         return False
16     num = abs(number)
17     digit_sum = sum(int(digit) for digit in str(num))
18     if digit_sum == 0:
19         return False
20     return num % digit_sum == 0
21
22 def main():
23     """
24     Handle user input and display whether the number
25     is a Harshad number.
26     """
27     number = int(input("Enter a number: "))
28     result = is_harshad(number)
29     print(result)
30
31 if __name__ == "__main__":
32     main()

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "C:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/18-02-26.py"
Enter a number: 60
True
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>
```

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

```
18-02-26.py>...
1 def count_trailing_zeros(n):
2     """
3     counting multiples of 5 instead of computing the full factorial.
4     Parameters:
5     n (int): A non-negative integer.
6     Returns:
7     int: The number of trailing zeros in n!.
8     Raises:
9     ValueError: If n is negative.
10    """
11    if n < 0:
12        raise ValueError("Input must be a non-negative integer.")
13    count = 0
14    divisor = 5
15    while n // divisor > 0:
16        count += n // divisor
17        divisor *= 5
18    return count
19
20 def main():
21     """
22     Handle user input and display trailing zeros result.
23     """
24     n = int(input("Enter a number: "))
25     try:
26         result = count_trailing_zeros(n)
27         print("Trailing zeros:", result)
28     except ValueError as e:
29         print(e)
30
31 if __name__ == "__main__":
32     main()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding> & C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64\python.exe
AI Assist Coding\18-02-26.py
Enter a number: 5
Trailing zeros: 1
PS C:\Users\SRAVANI\OneDrive\Documents\AI Assist Coding> |
```

Task 13 (Collatz Sequence Generator – Test Case Design)

- Function: Generate Collatz sequence until reaching 1.
- Test Cases to Design:
- Normal: $6 \rightarrow [6, 3, 10, 5, 16, 8, 4, 2, 1]$
- Edge: $1 \rightarrow [1]$
- Negative: -5
- Large: 27 (well-known long sequence)
- Requirement: Validate correctness with pytest.

Explanation:

We need to write a function that:

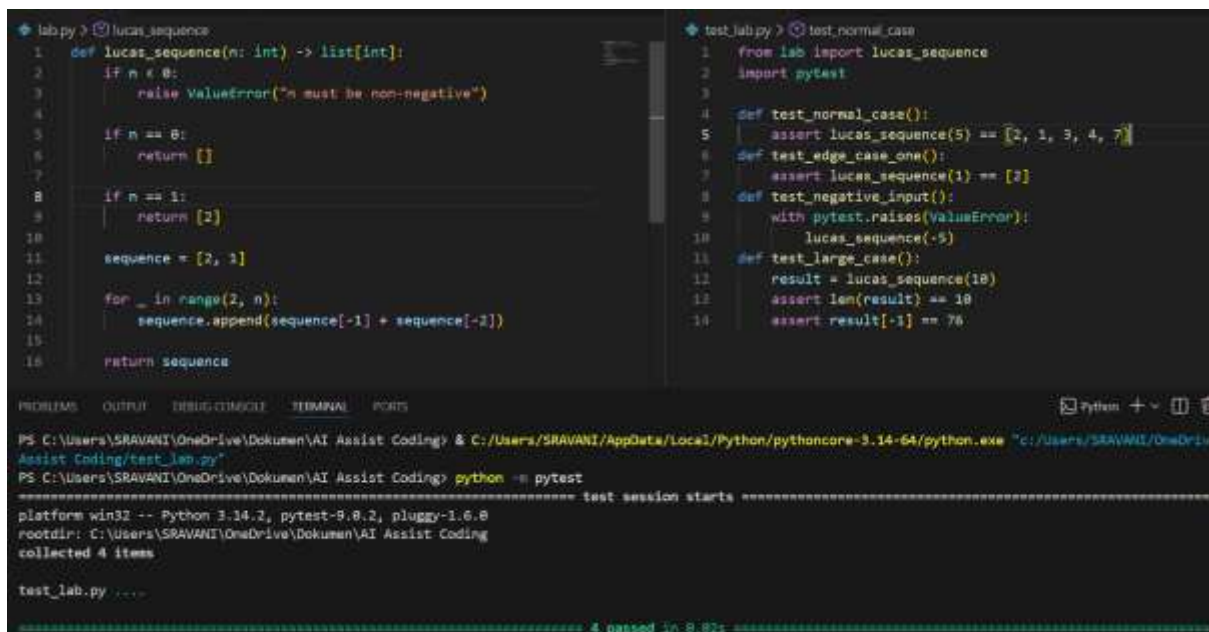
- Takes an integer n as input.
- Generates the Collatz sequence (also called the $3n+1$ sequence).
- The rules are:

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

- **Function:** Generate Lucas sequence up to n terms.

(Starts with 2,1, then $F_n = F_{n-1} + F_{n-2}$)

- Test Cases to Design:
- Normal: $5 \rightarrow [2, 1, 3, 4, 7]$
- Edge: $1 \rightarrow [2]$
- Negative: $-5 \rightarrow \text{Error}$
- Large: 10 (last element = 76).
- Requirement: Validate correctness with pytest.



```
lab.py > lucas_sequence
1 def lucas_sequence(n: int) -> list[int]:
2     if n < 0:
3         raise ValueError("n must be non-negative")
4
5     if n == 0:
6         return []
7
8     if n == 1:
9         return [2]
10
11    sequence = [2, 1]
12
13    for _ in range(2, n):
14        sequence.append(sequence[-1] + sequence[-2])
15
16    return sequence

test_lab.py > test_normal_case
1 from lab import lucas_sequence
2 import pytest
3
4 def test_normal_case():
5     assert lucas_sequence(5) == [2, 1, 3, 4, 7]
6
7 def test_edge_case_one():
8     assert lucas_sequence(1) == [2]
9
10 def test_negative_input():
11     with pytest.raises(ValueError):
12         lucas_sequence(-5)
13
14 def test_large_case():
15     result = lucas_sequence(10)
16     assert len(result) == 10
17     assert result[-1] == 76

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/Users/SRAVANI/OneDrive/Dokumen\AI Assist Coding/test_lab.py"
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> python -m pytest
===== test session starts =====
platform win32 -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding
collected 4 items

test_lab.py ....

===== 4 passed in 0.02s =====
```

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

- Function: Count vowels and consonants in string.
- Test Cases to Design:
- Normal: "hello" $\rightarrow (2,3)$
- Edge: "" $\rightarrow (0,0)$
- Only vowels: "aeiou" $\rightarrow (5,0)$
- Large: Long text
- Requirement: Validate correctness with pytest.

```
lab.py • test_lab.py •
lab.py > ..
1 # lab.py
2
3 def count_vowels_consonants(text: str) -> tuple[int, int]:
4     vowels = set('aeiouAEIOU')
5     vowel_count = 0
6     consonant_count = 0
7
8     for char in text:
9         if char.isalpha():
10             if char in vowels:
11                 vowel_count += 1
12             else:
13                 consonant_count += 1
14
15     return vowel_count, consonant_count
16

test_lab.py •
test_lab.py > test_only_vowels
1 from lab import count_vowels_consonants
2
3 def test_normal_case():
4     assert count_vowels_consonants("hello") == (2, 3)
5
6 def test_empty_string():
7     assert count_vowels_consonants("") == (0, 0)
8
9 def test_only_vowels():
10    assert count_vowels_consonants("aeiou") == (5, 0)
11
12 def test_large_text():
13    text = "This is a very long text " * 10000
14    vowels, consonants = count_vowels_consonants(text)
15    assert vowels > 0
16    assert consonants > 0
17    assert vowels + consonants == sum(c.isalpha() for c in text)

PROBLEMS OUTPUT DEBUG-CONSOLE TERMINAL PLOTS
Python + - - - - -

PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:\Users\SRAVANI\AppData\Local\Python\pythoncore-3.14-64/python.exe "c:/Users/SRAVANI/OneDrive/Dokumen/AI Assist Coding/test_lab.py"
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> python -m pytest
===== test session starts =====
platform win32 -- Python 3.14.2, pytest-9.0.2, pluggy-1.6.0
rootdir: C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding
collected 4 items

test_lab.py ..... [tests]

===== 4 passed in 0.83s =====
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding>
```