

# **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

```

❸ 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
21 # Example usage:
22 rectangles = [(5, 10), (7, 12), (10, 15)]
23 for length, width in rectangles:
24     area, perimeter = calculate_rectangle_properties(length, width)
25     print(f"Area of Rectangle: {area}")
26     print(f"Perimeter of Rectangle: {perimeter}")
27

```

```

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):
2     price (int or float): The original price of the product.
3     Returns:
4     float: The total price after adding 18% tax.
5     ....
6     tax = price * 0.18
7     total = price + tax
8     return total
9
10 def main():
11     ....
12
13     Main function to calculate and print total prices
14     for multiple product values.
15     Returns:
16     None
17     ....
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
xe "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  $\geq 90$  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  
 9      def calculate_grade(self, marks):  
10          """  
11              Determine the grade for a given mark.  
12              Parameters:  
13                  marks (int or float): The student's marks (assumed between 0 and 100).  
14              Returns:  
15                  str: The grade corresponding to the marks based on the grading rules.  
16          """  
17  
18          if marks >= 90 and marks <= 100:  
19              return "Grade A"  
20          elif marks >= 80:  
21              return "Grade B"  
22          elif marks >= 70:  
23              return "Grade C"  
24          elif marks >= 40:  
25              return "Grade D"  
26          elif marks >= 0:  
27              return "Fail"  
28  
29      def main():  
30          """  
31              Main function to demonstrate reusable grade calculation.  
32          """  
33  
34          calculator = GradeCalculator()  
35          marks_list = [85, 72, 95, 38]  
36          for marks in marks_list:  
37              grade = calculator.calculate_grade(marks)  
38              print(f"Marks: {marks} -> {grade}")  
39  
40  if __name__ == "__main__":  
41      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
 9 def calculate_square(num):
10     """
11         Calculate the square of a number.
12         Parameters:
13             num (int or float): the number to be squared.
14         Returns:
15             int or float: the square of the given number.
16         """
17     return num * num
18
19 def display_result(square):
20     """
21         Display the square result.
22         Parameters:
23             square (int or float): The calculated square value.
24         Returns:
25             None
26         """
27     print("Square:", square)
28
29 def main():
30     """
31         Main function to coordinate input, processing,
32         and output operations.
33         """
34     number = get_input()
35     square = calculate_square(number)
36     display_result(square)
37 if __name__ == "__main__":
38     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.

```
13.py > ..  
1  class EmployeeSalaryCalculator:  
2    def calculate_tax(self):  
3      """  
4          return self.salary * self.tax_rate  
5      """  
6      # calculate the net salary after tax deduction.  
7      # Returns:  
8      # float: The net salary.  
9      #  
10     tax = self.calculate_tax()  
11     return self.salary - tax  
12  def display_net_salary(self):  
13      """  
14          display the net salary.  
15          Returns:  
16          None  
17      """  
18      net_salary = self.calculate_net_salary()  
19      print("Net Salary:", net_salary)  
20  def main():  
21      """  
22          Main function to demonstrate salary calculation.  
23      """  
24      employee = EmployeeSalaryCalculator(50000)  
25      employee.display_net_salary()  
26  if __name__ == "__main__":  
27      main()  
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL FODS  
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "c:/users/srava/  
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

The screenshot shows a code editor with Python code named `13.py`. The code defines a function `check_access` that takes a set of users and a username, returning `True` if the user exists. It also contains a `main` function that reads a username from input, checks it against the set, and prints "Access Granted" or "Access Denied". The code is annotated with docstrings and comments explaining the logic and data structures used.

```
1  #!/usr/bin/env python3
2  def check_access(users_set, username):
3      """
4          Check whether the given username exists in the system.
5          Parameters:
6              users_set (set): A set of valid usernames.
7              username (str): The username to verify.
8          Returns:
9              bool: True if user exists, otherwise False.
10         """
11     return username in users_set
12
13 def main():
14     """
15         Main function to handle user input and access verification.
16         """
17     users = {"admin", "guest", "editor", "viewer"} # Using set for O(1) lookup
18     name = input("Enter username: ")
19     if check_access(users, name):
20         print("Access Granted")
21     else:
22         print("Access Denied")
23 if __name__ == "__main__":
24     main()
```

Below the code editor is a terminal window showing the execution of the script and its output. The terminal shows the command `python 13.py`, the prompt `Enter username:`, the user input `sravani`, and the resulting output `Access Denied`.

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

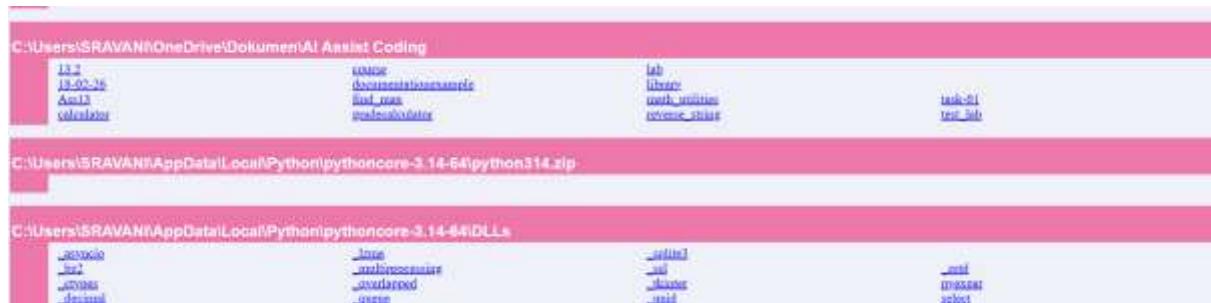
```
else:  
    print("Book already exists.")  
  
# Adding second book (duplicate logic)  
title = "AI Fundamentals"  
author = "Jane Smith"  
isbn = "102"  
  
if isbn not in library_db:  
    library_db[isbn] = {"title": title, "author": author}  
    print("Book added successfully.")  
  
else:  
    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
25  def remove_book(isbn):
26      ...
27
28      Removes a book from the library database.
29
30      Parameters:
31          isbn (str): ISBN of the book to remove.
32
33      Returns:
34          str: Success or not found message.
35
36      if isbn in library_db:
37          del library_db[isbn]
38          return "Book removed successfully."
39      return "Book not found."
40
41
42
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ▾

```

WRITER 33.html  
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\AI Assist Coding\13.py"  
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> python -m pydoc -p 1432  
Server ready at: http://localhost:1432/  
Server commands: [b]rowser, [q]uit  
server> http://localhost:1432/



## 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
```

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

At 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\SRAWNIE\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.

```
1  def get_zodiac(year):
2      """Given a year, return the Chinese zodiac sign as a string.
3
4      Returns:
5          str: The corresponding Chinese zodiac sign.
6
7      """
8
9      zodiac_cycle = [
10         "Monkey", # 0
11         "Rooster", # 1
12         "Dog", # 2
13         "Pig", # 3
14         "Rat", # 4
15         "Ox", # 5
16         "Tiger", # 6
17         "Rabbit", # 7
18         "Dragon", # 8
19         "Snake", # 9
20         "Horse", # 10
21         "Goat" # 11
22     ]
23     return zodiac_cycle[year % 12]
24
25 def main():
26     """
27     Handle user input and display the zodiac sign.
28     """
29     year = int(input("Enter a year: "))
30     zodiac_sign = get_zodiac(year)
31     print(zodiac_sign)
32
33 if __name__ == "__main__":
34
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: 2028
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:

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

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

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

```

❶ 18-02-26.py >-
❷ def count_trailing_zeros(n):
❸     """Counting trailing zeros in n instead of computing the full factorial.
❹     Parameters:
❺     n (int): A non-negative integer.
❻     Returns:
❼     int: The number of trailing zeros in n.
❽     Raises:
❾     ValueError: If n is negative.
❿     """
❿
❿ if n < 0:
❿     raise ValueError("Input must be a non-negative integer.")
❿ count = 0
❿ divisor = 5
❿ while n // divisor > 0:
❿     count += n // divisor
❿     divisor *= 5
❿ return count
❿
❿ def main():
❿     """
❿     Handle user input and display trailing zeros result.
❿     """
❿
❿     n = int(input("Enter a number: "))
❿     try:
❿         result = count_trailing_zeros(n)
❿         print("Trailing zeros:", result)
❿     except ValueError as e:
❿         print(e)
❿
❿ if __name__ == "__main__":
❿
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 number: 5
Trailing zeros: 1
PS C:\Users\SRAVANI\OneDrive\Dokumen\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:

- o If n is even  $\rightarrow$  next =  $n / 2$ .
  - o If n is odd  $\rightarrow$  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]

```
def collatz_sequence(n: int) -> list[int]:
    if n < 0:
        raise ValueError("n must be a positive integer")
    sequence = [n]
    while n != 1:
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
        sequence.append(n)
    return sequence

Example usage
if __name__ == "__main__":
    print(collatz_sequence(6)) # Output: [6, 3, 10, 5, 16, 8, 4, 2, 1]
    print(collatz_sequence(1)) # Output: [1]
    print(collatz_sequence(27)) # Output: A long sequence starts ...

# test_lab.py -> test_large_case_27
from lab import collatz_sequence
import pytest
def test_normal_case():
    assert collatz_sequence(6) == [6, 3, 10, 5, 16, 8, 4, 2, 1]

def test_edge_case_one():
    assert collatz_sequence(1) == [1]

def test_negative_input():
    with pytest.raises(ValueError):
        collatz_sequence(-5)

def test_large_case_27():
    result = collatz_sequence(27)
    # Basic correctness checks
    assert result[0] == 27
    assert result[-1] == 1
    assert len(result) > 100 # 27 produces a long sequence (200+ terms)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\SRAVANI\OneDrive\Dokument\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14-64/python.exe "C:/Users/SRAVANI/OneDrive/Dokument/AI Assist Coding/Assist_Coding/test_lab.py"
PS C:\Users\SRAVANI\OneDrive\Dokument\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\Dokument\AI Assist Coding
collected 4 items

test_lab.py ...
```

4 passed in 0.06s

```
PS C:\Users\SRAVANI\OneDrive\Dokument\AI Assist Coding>
```

## Task 14 (Lucas Number Sequence – Test Case Design)

- 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 → [2, 1, 3, 4, 7]
- Edge: 1 → [2]
- Negative: -5 → Error
- Large: 10 (last element = 76).
- Requirement: Validate correctness with pytest.

The screenshot shows a code editor with two files open:

- lucas\_sequence.py:** A Python function that generates the Lucas sequence. It handles non-negative integers up to 18. For negative inputs, it raises a ValueError. For n=0, it returns an empty list. For n=1, it returns [2]. For n>1, it uses a loop to build the sequence from [2, 1] up to the nth element.
- test\_lab.py:** A pytest test module containing four test functions: `test_normal_case`, `test_edge_case_one`, `test_negative_input`, and `test_large_case`. Each test asserts the correct output or raises the expected error for specific input values.

The terminal at the bottom shows the command `python -m pytest` being run, which starts a test session and collects 4 items, then runs the tests for both files, resulting in 4 passed tests.

### 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.

```
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
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
371
```

```
test_lab.py
1 test_lab.py > @test_only_vowels
2     from lab import count_vowels_consonants
3
4     def test_normal_case():
5         assert count_vowels_consonants("hello") == (2, 3)
6
7     def test_empty_string():
8         assert count_vowels_consonants("") == (0, 0)
9
10    def test_only_vowels():
11        assert count_vowels_consonants("aeiou") == (5, 0)
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 POINTS Python + -

```
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> & C:/Users/SRAVANI/AppData/Local/Python/pythoncore-3.14.2-64/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 ... (100%)
===== 4 passed in 0.89s =====
PS C:\Users\SRAVANI\OneDrive\Dokumen\AI Assist Coding> []
```