

AI Assisted Coding Assignment-13.1



Task Description #1 (Refactoring – Removing Code Duplication)

- Task: Use AI to refactor a given Python script that contains multiple repeated code blocks.
- Instructions:
 - Prompt AI to identify duplicate logic and replace it with functions or classes.
 - Ensure the refactored code maintains the same output.
 - 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:
 - Refactored code with a reusable function and no duplication.

Well documented code

The screenshot shows a code editor window with the following details:

- File Tabs:** .py, ass_9_5.py, Ass_10.2.py, Ass_13.1.py 1 (highlighted), String_Ass1.py
- Code Content:**

```
5  #print("Area of Rectangle:", 7 * 12)
6  #print("Perimeter of Rectangle:", 2 * (7 + 12))
7  #print("Area of Rectangle:", 10 * 15)
8  #print("Perimeter of Re
9 #Prompt:Identify duplic
10 def calculate_rectangle_area(length, width):
    """
    Calculate the area of a rectangle.

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

    Returns:
    int or float: The area of the rectangle (Length * width).

    Example:
    >>> calculate_rectangle_area(5, 10)
    50
    """
    return Length * width
11
```
- Toolbars and Status:** Includes icons for file operations, search, and tabs. A status bar at the bottom right shows "Accept Tab" and "Accept Word [Ctrl + RightArrow]".

```

File Edit Selection View Go ...
Ass_13.1.py X
Ass_9_5.py Ass_10.2.py Ass_13.1.py String_Ass1.py Ass_4.5.py Ass_4.5(2)

26 def calculate_rectangle_perimeter(length, width):
27     """
28     Parameters:
29         length (int or float): The length of the rectangle.
30         width (int or float): The width of the rectangle.
31     Returns:
32         int or float: The perimeter of the rectangle ( $2 * (length + width)$ ).
33     Example:
34     >>> calculate_rectangle_perimeter(5, 10)
35     30
36     """
37     return 2 * (length + width)
38
39 # Example usage
40 lengths = [5, 7, 10]
41 widths = [10, 12, 15]
42 for length, width in zip(lengths, widths):
43     area = calculate_rectangle_area(length, width)
44     perimeter = calculate_rectangle_perimeter(length, width)
45     print(f"Area of Rectangle: {area}")
46     print(f"Perimeter of Rectangle: {perimeter}")

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python/OneDrive/Desktop/A.I.AC/Ass_13.1.py
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\hruth\OneDrive\Desktop\A.I.AC>

```



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:
 - Identify repeated or related logic and extract it into reusable functions.
 - 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:
 - Code with a function calculate_total(price) that can be reused for multiple price inputs.
 - Well documented code

```
py ass_9_5.py Ass_10.2.py Ass_13.1.py String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py Ass_5.5.py
Ass_13.1.py > ...
53 #total = price + tax
54 #print("Total Price:", total)
55 #price = 500
56 #tax = price * 0.18
57 #total = price + tax
58 #print("Total Price:", total)
59 #Prompt:Execute a Well documented Code with a function calculate_total(price) that can be reused for multiple price inputs.
60
61 def calculate_total(price):
    """
    Calculate the total price including tax.

    Parameters:
    price (int or float): The base price.

    Returns:
    float: The total price including tax (18% tax rate).

    Example:
    >>> calculate_total(250)
    295.0
    """
    tax = price * 0.18
    return price + tax

# Example usage
prices = [250, 500]
for price in prices:
    total = calculate_total(price)
    print(f"Total Price: {total}")


```

```
File Edit Selection View Go ... ← → Q AIAC 08 W
?py ass_9_5.py Ass_10.2.py Ass_13.1.py String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py Ass_5.5.py
Ass_13.1.py > calculate_total
58 #print('Total Price:', total)
59 #Prompt:Execute a Well documented Code with a function calculate_total(price) that can be reused for multiple price inputs.
60 def calculate_total(price):
61     """
62     Calculate the total price including tax.
63
64     Parameters:
65     price (int or float): The original price of the item.
66
67     Returns:
68     int or float: The total price after adding 18% tax.
69
70     Example:
71     >>> calculate_total(250)
72     295.0
73     """
74     tax = price * 0.18 # Calculate tax as 18% of the price
75     total = price + tax # Add tax to the original price to get the total
76     return total
77 # Example usage
78 prices = [250, 500]
79 for price in prices:
80     total_price = calculate_total(price)
81     print(f"Total Price: {total_price}")
82
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC
Total Price: 295.0
Total Price: 590.0
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```



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:

- Accept marks as a parameter.
- Return the corresponding grade as a string.
- 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.

1. Include proper docstrings for:

- The class
- The method (with parameter and return descriptions)

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

• Given code:

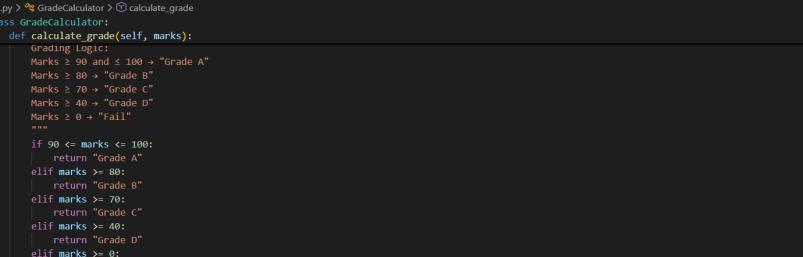
```
marks = 85
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
marks = 72
if marks >= 90:
    print("Grade A")
elif marks >= 75:
    print("Grade B")
else:
    print("Grade C")
```

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.

```
File Edit Selection view ... Q A.I.AC  
1.py ass_9.py Ass_10.2.py Ass_13.1.py ● String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py  
Ass_13.1.py > ...  
115 #The method (with parameter and return descriptions)  
116 #The method must be reusable and called multiple times without rewriting conditional logic.  
117 class GradeCalculator:  
    """  
        A class to calculate the grade based on student marks.  
  
    Methods:  
        calculate_grade(marks): Accepts marks as input and returns the corresponding grade.  
        """  
    def calculate_grade(self, marks):  
        """  
            Calculate the grade based on the given marks.  
  
        Parameters:  
            marks (int): The marks obtained by the student (0 to 100).  
  
        Returns:  
            str: The grade corresponding to the marks.  
  
        Grading Logic:  
        Marks ≥ 90 and ≤ 100 → "Grade A"  
        Marks ≥ 80 → "Grade B"  
        Marks ≥ 70 → "Grade C"  
        Marks ≥ 40 → "Grade D"  
        Marks ≥ 0 → "Fail"  
        """  
        if 90 <= marks <= 100:  
            return "Grade A"  
        elif marks >= 80:  
            return "Grade B"  
        elif marks >= 70:  
            return "Grade C"  
        elif marks >= 40:  
            return "Grade D"  
        elif marks >= 0:  
            return "Fail."  
    """  
    A class to calculate the grade based on student marks.  
  
Methods:  
calculate_grade(marks): Accepts marks as input and returns the corresponding grade.  
"""  
def calculate_grade(self, marks):  
    """  
Calculate the grade based on the given marks.  
  
Parameters:  
marks (int): The marks obtained by the student (0 to 100).  
  
Returns:  
str: The grade corresponding to the marks.  
  
Grading Logic:  
Marks ≥ 90 and ≤ 100 → "Grade A"  
Marks ≥ 80 → "Grade B"  
Marks ≥ 70 → "Grade C"  
Marks ≥ 40 → "Grade D"  
Marks ≥ 0 → "Fail"  
"""  
if 90 <= marks <= 100:  
    return "Grade A"  
elif marks >= 80:  
    return "Grade B"  
elif marks >= 70:  
    return "Grade C"  
elif marks >= 40:  
    return "Grade D"  
elif marks >= 0:  
    return "Fail."  
    """
```



```
File Edit Selection View ... ← → Q A.I.AC

Ass_13.1.py Ass_9.5.py Ass_10.2.py Ass_13.1.py X String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py Ass_5.5.py Ass_6.5.py Ass_7.5.py

1. py

117 class GradeCalculator:
118     def calculate_grade(self, marks):
119         """
120             Grading Logic:
121             Marks ≥ 90 and ≤ 100 → "Grade A"
122             Marks ≥ 80 → "Grade B"
123             Marks ≥ 70 → "Grade C"
124             Marks ≥ 40 → "Grade D"
125             Marks ≥ 0 → "Fail"
126         """
127
128         if 90 <= marks <= 100:
129             return "Grade A"
130         elif marks >= 80:
131             return "Grade B"
132         elif marks >= 70:
133             return "Grade C"
134         elif marks >= 40:
135             return "Grade D"
136         elif marks >= 0:
137             return "Fail"
138         else:
139             raise ValueError("Marks should be between 0 and 100.")
140
141 # Example usage
142 grade_calculator = GradeCalculator()
143 marks_list = [85, 72, 95, 65, 30]
144 for marks in marks_list:
145     grade = grade_calculator.calculate_grade(marks)
146     print(f"Marks: {marks}, Grade: {grade}")

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/users/hruth/OneDrive/Desktop/A.I.AC/Ass_13.1.py
Marks: 85, Grade: Grade B
Marks: 72, Grade: Grade C
Marks: 95, Grade: Grade A
Marks: 65, Grade: Grade D
Marks: 30, Grade: Fail

```



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().

```
9  #Task Description #4 (Refactoring - Converting Procedural Code to Functions)
10 #Sample Legacy Code:
11 #num = int(input("Enter number: "))
12 #square = num * num
13 #print("Square:", square)
14 #Prompt:Generate a modular code with docstrings using functions like get_input(), calculate_square(), and display_result()
15 def get_input():
16     """
17         Prompt the user to enter a number and return it as an integer.
18
19         Returns:
20             int: The number entered by the user.
21         """
22     return int(input("Enter number: "))
23
24
25
```

```
File Edit Selection View ... ← → Q AI.AC
2.py ass_9.5.py Ass_10.2.py Ass_13.1.py X String_Ass1.py Ass_4.5.py Ass_4.5(2)
Ass_13.1.py > main
174 def calculate_square(num):
181     """
182     return num * num
183 def display_result(square):
184     """
185     Display the result of the square calculation.
186     Parameters:
187     square (int): The squared value to be displayed.
188     """
189     print("Square:", square)
190     # Main function to orchestrate the flow of the program
191 def main():
192     number = get_input() # Get user input
193     square = calculate_square(number) # Calculate the square of the input number
194     display_result(square) # Display the result
195     # Execute the main function
196     if __name__ == "__main__":
197         main()

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python
Marks: 30, Grade: Fail
Enter number: 1543
Square: 2380849
Ln 193, Col 50 Spaces: 4 UTF-8 CRLF {} Python 3.12.3 Python 3.12.3
```



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().

The screenshot shows a code editor with a dark theme. The file is named 'Ass_13.1.py'. The code defines a class 'EmployeeSalaryCalculator' with methods for calculating net salary after tax deduction. The code includes detailed docstrings for each method, specifying attributes, parameters, and return types. The code editor interface shows tabs for other files like 'ass_9.5.py', 'ass_10.2.py', etc., and status bars at the bottom showing line and column counts, encoding, and Python version.

```
!py ass_9.5.py Ass_10.2.py Ass_13.1.py ● String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py Ass_5.5.py
Ass_13.1.py > ...
205 #Prompt: Refactor procedural code into a class-based design like EmployeeSalaryCalculator with methods and attributes.
206 class EmployeeSalaryCalculator:
    """
    A class to calculate the net salary of an employee after tax deduction.

    Attributes:
        salary (float): The gross salary of the employee.

    Methods:
        calculate_net_salary(): Calculates and returns the net salary after tax deduction.
    """
    def __init__(self, salary):
        """
        Initialize the EmployeeSalaryCalculator with the given salary.

        Parameters:
            salary (float): The gross salary of the employee.
        """
        self.salary = salary
    def calculate_net_salary(self):
        """
        Calculate the net salary by deducting 20% tax from the gross salary.

        Returns:
            float: The net salary after tax deduction.
        """
        tax = self.salary * 0.2 # Calculate tax as 20% of the gross salary
        net_salary = self.salary - tax # Subtract tax from gross salary to get net salary
        return net_salary
207
```

```

File Edit Selection View Go ...
Q A.I.AC
.py ass_9.py Ass_10.2.py Ass_13.1.py X String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py

206     class EmployeeSalaryCalculator:
214         calculate_net_salary(): Calculates and returns the net salary after tax deduction.
215         """
216     def __init__(self, salary):
217         """
218             Initialize the EmployeeSalaryCalculator with the given salary.
219
220             Parameters:
221             salary (float): The gross salary of the employee.
222             """
223         self.salary = salary
224     def calculate_net_salary(self):
225         """
226             Calculate the net salary by deducting 20% tax from the gross salary.
227
228             Returns:
229             float: The net salary after tax deduction.
230             """
231         tax = self.salary * 0.2 # Calculate tax as 20% of the gross salary
232         net_salary = self.salary - tax # Subtract tax from gross salary to get net salary
233         return net_salary
234
235     # Example usage
236     employee_salary_calculator = EmployeeSalaryCalculator(50000) # Create an instance with a salary of 50000
237     net_salary = employee_salary_calculator.calculate_net_salary() # Calculate the net salary
238     print(net_salary) # Expected output: 40000.0

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/C:/Users/hruth/OneDrive/Desktop/A.I.AC>



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 dark-themed code editor with a sidebar containing various icons. The main area displays Python code for a function named `check_user_access`. The code includes docstrings with parameter descriptions, return types, and time complexity analysis.

```
242     name = input("Enter username: ")
243     found = False
244     for u in users:
245         if u == name:
246             found = True
247     print("Access Granted" if found else "Access Denied")
248     #Prompt:Refactor inefficient linear searches using appropriate data structures.
249     #Output:Use of sets or dictionaries with complexity justification
250     #Focus Areas:Time complexity and Data structure choice
251     def check_user_access(username, user_set):
252         """
253             Check if the given username exists in the user set and return access status.
254
255             Parameters:
256             username (str): The username to be checked.
257             user_set (set): A set of valid usernames for access control.
258
259             Returns:
260             str: "Access Granted" if the username is found, otherwise "Access Denied".
261
262             Time Complexity:
263             The use of a set allows for O(1) average time complexity for membership checks,
264             making it more efficient than a list which has O(n) time complexity for the same operation.
265         """
266         return "Access Granted" if username in user_set else "Access Denied"
```

The screenshot shows the same code editor with the file `Ass_13.1.py` open. The terminal tab at the bottom shows the execution of the script and its output. The user enters "admin" and receives the response "Access Granted".

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
```

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/Ass_13.1.py
Enter username: admin
Access Granted
PS C:/Users/hruth/OneDrive/Desktop/A.I.AC>
```



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:

- add_book(title, author, isbn)
- remove_book(isbn)
- 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:
```

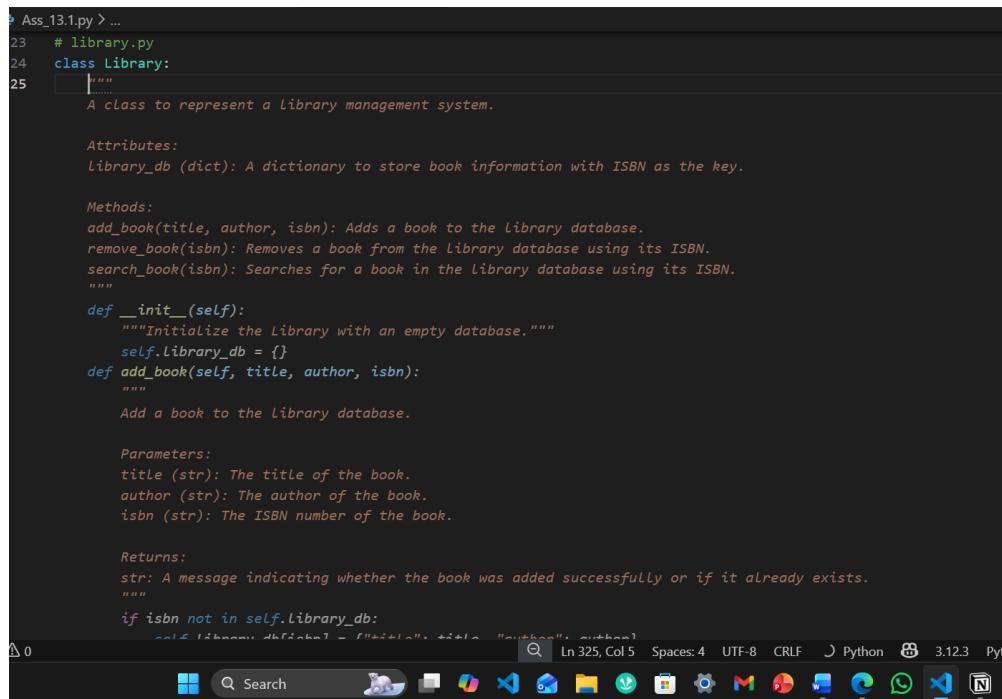
```

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.")

```



```

# Ass_13.1.py > ...
23  # library.py
24  class Library:
25      """
A class to represent a Library management system.

Attributes:
library_db (dict): A dictionary to store book information with ISBN as the key.

Methods:
add_book(title, author, isbn): Adds a book to the Library database.
remove_book(isbn): Removes a book from the Library database using its ISBN.
search_book(isbn): Searches for a book in the Library database using its ISBN.
"""
def __init__(self):
    """Initialize the Library with an empty database."""
    self.library_db = {}
def add_book(self, title, author, isbn):
    """
Add a book to the library database.

Parameters:
title (str): The title of the book.
author (str): The author of the book.
isbn (str): The ISBN number of the book.

Returns:
str: A message indicating whether the book was added successfully or if it already exists.
"""
    if isbn not in self.library_db:
        self.library_db[isbn] = {"title": title, "author": author}
Ln 325, Col 5 Spaces: 4 UTF-8 CRLF Python 3.12.3 Pyt

```

```

File Edit Selection View Go ... ← → Q A.I.AC
.py ass_9_5.py Ass_10.2.py Ass_13.1.py X String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py

324     class Library:
325         def search_book(self, isbn):
326             """Search for a book in the library database using its ISBN.
327
328             Parameters:
329                 isbn (str): The ISBN number of the book to be searched.
330
331             Returns:
332                 str: A message indicating whether the book was found along with its details or if it was not found.
333
334             """
335             if isbn in self.library_db:
336                 return f"Book Found: {self.library_db[isbn]}"
337             else:
338                 return "Book not found."
339
340     # Example usage
341     library = Library() # Create an instance of the Library class
342     print(library.add_book("Python Basics", "John Doe", "101")) # Add first book
343     print(library.add_book("AI Fundamentals", "Jane Smith", "102")) # Add second book
344     print(library.search_book("101")) # Search for the first book
345     print(library.remove_book("101")) # Remove the first book
346     print(library.search_book("101")) # Search for the first book again after removal

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC>

Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
Help on module library:

```

NAME
    library - # library.py

CLASSES
    builtins.object
        Library

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

            Attributes:
                library_db (dict): A dictionary to store book information with ISBN as the key.

            Methods:
                add_book(title, author, isbn): Adds a book to the library database.
                remove_book(isbn): Removes a book from the library database using its ISBN.
                search_book(isbn): Searches for a book in the library database using its ISBN.

            Methods defined here:

            __init__(self)
            -- More -- 

```

```

    Initialize the Library with an empty database.

add_book(self, title, author, isbn)
    Add a book to the library database.

    Parameters:
        title (str): The title of the book.
        author (str): The author of the book.
        isbn (str): The ISBN number of the book.

    Returns:
        str: A message indicating whether the book was added successfully or if it already exists.

remove_book(self, isbn)
    Remove a book from the library database using its ISBN.

    Parameters:
        isbn (str): The ISBN number of the book to be removed.

    Returns:
        str: A message indicating whether the book was removed successfully or if it was not found.

search_book(self, isbn)
    Search for a book in the library database using its ISBN.

    Parameters:
        isbn (str): The ISBN number of the book to be searched.

    Returns:
        str: A message indicating whether the book was found along with its details or if it was not found.

Data descriptors defined here:

__dict__

```

Ln 393, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.3

The screenshot shows the PyCharm IDE interface. On the left, there is a file tree containing several files: ASS-9.5(1543).pdf, ASS-10.2(1543).pdf, data.txt, labexam.py, library.html, library.py, String_Ass1.py, String.py, user_activity.log, and Week5_Documentatio... . Below the file tree are navigation buttons for 'OUTLINE' and 'TIMELINE'. The main code editor window displays the following Python code:

```

Data descriptors defined here:

__dict__
    dictionary for instance variables

__weakref__
    list of weak references to the object

DATA
    library = <library.Library object>

FILE
    c:\users\hruth\onedrive\desktop\A.I.AC\library.py

```

At the bottom of the code editor, there is a status bar showing the path 'PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>' and the file name 'library.py'. The bottom right corner of the screen shows the Windows taskbar with various pinned icons.

← ⌂ ⓘ 127.0.0.1:5500/library.html

Home - NetMirror Classroom - GDB o... Google Cisco Networking... DataCamp H

`remove_book(isbn)`: Removes a book from the library database using its ISBN.
`search_book(isbn)`: Searches for a book in the library database using its ISBN.

Methods defined here:

`__init__(self)`
Initialize the `Library` with an empty database.

`add_book(self, title, author, isbn)`
Add a book to the library database.

Parameters:
`title` (str): The title of the book.
`author` (str): The author of the book.
`isbn` (str): The ISBN number of the book.

Returns:
`str`: A message indicating whether the book was added successfully or if it already exists.

`remove_book(self, isbn)`
Remove a book from the library database using its ISBN.

Parameters:
`isbn` (str): The ISBN number of the book to be removed.

Returns:
`str`: A message indicating whether the book was removed successfully or if it was not found.

`search_book(self, isbn)`
Search for a book in the library database using its ISBN.

Parameters:
`isbn` (str): The ISBN number of the book to be searched.

Returns:
`str`: A message indicating whether the book was found along with its details or if it was not found.

Data descriptors defined here:

`__dict__`
dictionary for instance variables

`__weakref__`
list of weak references to the object

Data
`library = <library.Library object>`





Task 8– Fibonacci Generator.

Write a program to generate Fibonacci series up to n.

The initial code has:

- Global variables.
- Inefficient loop.
- No functions or modularity.

Task for Students:

- Refactor into a clean reusable function (generate_fibonacci).
- Add docstrings and test cases.
- Compare AI-refactored vs original.

Bad Code Version:

```
# fibonacci bad version
n=int(input("Enter limit: "))
a=0
b=1
print(a)
print(b)
for i in range(2,n):
    c=a+b
    print(c)
    a=b
    b=c
```

```
Ass_13.1.py > ...
400     #print(b)
401     #for i in range(2,n):
402     #    c=a+b
403     #    print(c)
404     #    a=b
405     #    b=c
406     #prompt:Refactor into a clean reusable function (generate_fibonacci).Add docstrings and test cases.Compare AI-refactored vs original.
407 def generate_fibonacci(n):
        """
        Generate a list of Fibonacci numbers up to the nth number.

        Parameters:
        n (int): The number of Fibonacci numbers to generate.

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

        Example:
        >>> generate_fibonacci(10)
        [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
        """
        if n <= 0:
            return []
        elif n == 1:
            return [0]
        elif n == 2:
            return [0, 1]
        fib_sequence = [0, 1]
        for i in range(2, n):
            next_fib = fib_sequence[i-1] + fib_sequence[i-2]
            fib_sequence.append(next_fib)

        return fib_sequence
408
409
```

```

❸ Ass_13.1.py > ...
405 # D=C
406 #prompt:Refactor into a clean reusable function (generate_fibonacci).Add docstrings and test cases.Compare AI-refactored vs original.
407 def generate_fibonacci(n):
408     """
409         Generate a list of Fibonacci numbers up to the nth number.
410
411     Parameters:
412         n (int): The number of Fibonacci numbers to generate.
413
414     Returns:
415         list: A list containing the Fibonacci sequence up to the nth number.
416
417     Example:
418     >>> generate_fibonacci(10)
419     [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
420     """
421     if n <= 0:
422         return []
423     elif n == 1:
424         return [0]
425     elif n == 2:
426         return [0, 1]
427
428     fib_sequence = [0, 1]
429     for i in range(2, n):
430         next_fib = fib_sequence[i-1] + fib_sequence[i-2]
431         fib_sequence.append(next_fib)
432
433     return fib_sequence
434 # Example usage
435 print(generate_fibonacci(10)) # Expected output: [0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
436

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
[0, 1, 1, 2, 3, 5, 8, 13, 21, 34]
PS C:\Users\hruthi\OneDrive\Desktop\A.I.AC]
```



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")
```

ass_9_5.py Ass_10.2.py Ass_13.1.py library.py library.html String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py

```
455 #Prompt: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.
456 def is_prime(n):
    """
    Check if a number is prime.

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

    Returns:
    bool: True if the number is prime, False otherwise.

    Example:
    >>> is_prime(11)
    True
    >>> is_prime(12)
    False
    """
    if n <= 1:
        return False
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return False
    return True
```

File Edit Selection View Go ... A.I.AC

ass_9_5.py Ass_10.2.py Ass_13.1.py library.py library.html String_Ass1.py

```
Ass_13.1.py
```

```
478 def is_twin_prime(p1, p2):
489     >>> is_twin_prime(12, 14)
490     False
491     """
492     return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
493 def generate_twin_primes(start, end):
494     """
495     Generate a list of twin primes within a given range.
496     Parameters:
497     start (int): The starting number of the range.
498     end (int): The ending number of the range.
499     Returns:
500     list: A list of tuples, each containing a pair of twin primes.
501     Example:
502     >>> generate_twin_primes(10, 30)
503     [(11, 13), (17, 19), (29, 31)]
504     """
505     twin_primes = []
506     for num in range(start, end - 1):
507         if is_twin_prime(num, num + 2):
508             twin_primes.append((num, num + 2))
509     return twin_primes
510 # Example usage
511 print(generate_twin_primes(10, 30)) # Expected output: [(11, 13), (17, 19)]
```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.
[(11, 13), (17, 19)]
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> []
```



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.

The screenshot shows a code editor window with a dark theme. The file is named `Ass_13.1.py`. The code defines a function `get_zodiac` that returns the Chinese Zodiac sign for a given year. The code includes comments explaining the purpose, parameters, return value, and examples of usage. It also defines a list of zodiac signs.

```
543     #Separate input handling from logic.
544     #Improve readability and maintainability.
545     #Ensure output remains correct.
546 def get_zodiac(year):
    """
    Get the Chinese Zodiac sign for a given year.

    Parameters:
    year (int): The year for which to determine the Chinese Zodiac sign.

    Returns:
    str: The Chinese Zodiac sign corresponding to the given year.

    Example:
    >>> get_zodiac(2020)
    'Rat'
    >>> get_zodiac(2021)
    'Ox'
    """
    zodiac_signs = [
        "Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox",
        "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Goat"
    ]
    return zodiac_signs[year % 12]
```

The screenshot shows a code editor interface with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, and a search bar labeled 'A.I.AC'. Below the menu is a tab bar with several files: ass_9_5.py, Ass_10.2.py, Ass_13.1.py (which is the active file), library.py, library.html, and String_Ass1.py. The main code area contains the following Python script:

```
546 def get_zodiac(year):
547     """
548     Parameters:
549     year (int): The year for which to determine the Chinese Zodiac sign.
550
551     Returns:
552     str: The Chinese Zodiac sign corresponding to the given year.
553
554     Example:
555     >>> get_zodiac(2020)
556     'Rat'
557     >>> get_zodiac(2021)
558     'Ox'
559     """
560
561     zodiac_signs = [
562         "Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox",
563         "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Goat"
564     ]
565     return zodiac_signs[year % 12]
566
567 # Example usage
568 year = int(input("Enter a year: ")) # Get user input for the year
569 zodiac_sign = get_zodiac(year) # Get the corresponding Chinese Zodiac sign
570 print(zodiac_sign) # Print the Zodiac sign
```

The terminal below the code shows the output of running the script:

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> Enter a year: 2026
Horse
```



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:**

- $18 \rightarrow 1 + 8 = 9 \rightarrow 18 \div 9 = 2 \checkmark$ (Harshad Number)
- $19 \rightarrow 1 + 9 = 10 \rightarrow 19 \div 10 \neq \text{integer} \times$ (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:
 - Accept an integer parameter.
 - Return True if the number is divisible by the sum of its digits.
 - Return False otherwise.
3. Separate user input from core logic.
4. Add proper docstrings.
5. Improve readability and maintainability.
6. Ensure the program handles edge cases (e.g., 0, negative numbers).

```
Ass_13.1.py > ...
593     #4. Add proper docstrings.
594     #5. Ensure the program handles edge cases (e.g., 0, negative numbers).
595
596 ✓  def is_harshad(number):
597     """
598         Check if a number is a Harshad (Niven) number.
599
600     Parameters:
601         number (int): The number to check.
602
603     Returns:
604         bool: True if the number is a Harshad number, False otherwise.
605
606     Example:
607     >>> is_harshad(18)
608     True
609     >>> is_harshad(19)
610     False
611     """
612
613     if number == 0:
614         return False
615     sum_digits = sum(int(digit) for digit in str(abs(number)))
616     if sum_digits == 0:
617         return False
618     return number % sum_digits == 0
619
620     # Example usage
621     number = int(input("Enter a number: "))
622     print(is_harshad(number))

Ln 596, Col 3  Spaces: 4  UTF-8
```

```
File Edit Selection View Go ...
Q A.I.AC

ass_9_5.py Ass_10.2.py Ass_13.1.py X library.py String_Ass1.py Ass_4.5.p

596 def is_harshad(number):
597     """
598         Check if a number is a Harshad (Niven) number.
599
600     Parameters:
601         number (int): The number to check.
602
603     Returns:
604         bool: True if the number is a Harshad number, False otherwise.
605
606     Example:
607     >>> is_harshad(18)
608     True
609     >>> is_harshad(19)
610     False
611     >>> is_harshad(0)
612     False
613     >>> is_harshad(-12)
614     False
615     >>> is_harshad(-12)
616     False
617     """
618
619     if number < 0:
620         return False # Harshad numbers are typically defined for non-negative integers
621     temp = abs(number) # Use absolute value to handle negative numbers
622     sum_digits = 0
623     while temp > 0:
624         digit = temp % 10
625         sum_digits += digit
626         temp //= 10
627     if sum_digits == 0:
628         return False # Avoid division by zero
629     return number % sum_digits == 0
630
631     # Example usage
632     num = int(input("Enter a number: ")) # Get user input for the number
633     result = is_harshad(num) # Check if the number is a Harshad number
634     print(result) # Print the result (True or False)

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

...
Enter a number: 18
True
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>

Ln 614, Col 10  Spaces: 4  CRLF  {} Python
```



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:
 - Accept a non-negative integer n .
 - 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).

```
ass_9_5.py Ass_10_2.py Ass_13.1.py library.py library.html String_Ass1.py Ass_4_3.py Ass_4_3(2).py Ass_13.1.py > ...
654     # Handle edge cases (e.g., negative numbers, zero).
655     def count_trailing_zeros(n):
656         """
657             Count the number of trailing zeros in the factorial of a given non-negative integer.
658
659             Parameters:
660                 n (int): A non-negative integer for which to count trailing zeros in n!.
661
662             Returns:
663                 int: The number of trailing zeros in n!.
664
665             The number of trailing zeros in n! can be found by counting the number of times 5 is a factor in the numbers from 1 to n, since 10 is made of 2 and 5, and there are usually more factors of 2 than 5 in a factorial.
666
667             Example:
668                 >>> count_trailing_zeros(100)
669                 24
670                 >>> count_trailing_zeros(0)
671                 0
672                 >>> count_trailing_zeros(-5)
673                 ValueError: Input must be a non-negative integer.
674
675             if n < 0:
676                 raise ValueError("Input must be a non-negative integer.")
677
678             count = 0
679             power_of_5 = 5
680             while power_of_5 <= n:
681                 count += n // power_of_5
682                 power_of_5 *= 5
683
684
685             return count
686
687             # Example usage
688             n = int(input("Enter a number: ")) # Get user input for the number
689             try:
690                 trailing_zeros = count_trailing_zeros(n) # Count the trailing zeros in n!
691                 print("Trailing zeros:", trailing_zeros) # Print the result
692             except ValueError as e:
693                 print(e) # Print the error message if input is invalid
```

```
ass_9_5.py Ass_10_2.py Ass_13.1.py X library.py library.html String_Ass1.py A
Ass_13.1.py > ...
655     def count_trailing_zeros(n):
656         """
657             Count the number of trailing zeros in the factorial of a given non-negative integer.
658
659             Parameters:
660                 n (int): A non-negative integer for which to count trailing zeros in n!.
661
662             Returns:
663                 int: The number of trailing zeros in n!.
664
665             The number of trailing zeros in n! can be found by counting the number of times 5 is a factor in the numbers from 1 to n, since 10 is made of 2 and 5, and there are usually more factors of 2 than 5 in a factorial.
666
667             Example:
668                 >>> count_trailing_zeros(100)
669                 24
670                 >>> count_trailing_zeros(0)
671                 0
672                 >>> count_trailing_zeros(-5)
673                 ValueError: Input must be a non-negative integer.
674
675             if n < 0:
676                 raise ValueError("Input must be a non-negative integer.")
677
678             count = 0
679             power_of_5 = 5
680             while power_of_5 <= n:
681                 count += n // power_of_5
682                 power_of_5 *= 5
683
684
685             return count
686
687             # Example usage
688             n = int(input("Enter a number: ")) # Get user input for the number
689             try:
690                 trailing_zeros = count_trailing_zeros(n) # Count the trailing zeros in n!
691                 print("Trailing zeros:", trailing_zeros) # Print the result
692             except ValueError as e:
693                 print(e) # Print the error message if input is invalid
```



Test Cases Design

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:
 - If n is even → next = n / 2.
 - 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]

```

ass_9_5.py Ass_10.2.py Ass_13.1.py ● library.py library.html String_Ass1.py Ass_4.5.py Ass_4
Ass_13.1.py > ...
720 def generate_collatz_sequence(n):
    """
    Generate the Collatz sequence (3n+1 sequence) for a given integer n until it reaches 1.

    Parameters:
    n (int): The starting integer for the Collatz sequence.

    Returns:
    List: A List containing the Collatz sequence starting from n and ending at 1.

    Example:
    >>> generate_collatz_sequence(6)
    [6, 3, 10, 5, 16, 8, 4, 2, 1]
    >>> generate_collatz_sequence(1)
    [1]
    >>> generate_collatz_sequence(-5)
    ValueError: Input must be a positive integer.
    >>> generate_collatz_sequence(27)
    [27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484,
     242, 121, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155,
     466, 233, 700, ... ,1] # (Long sequence truncated for brevity)
    """
    if n <= 0:
        raise ValueError("Input must be a positive integer.")

    sequence = []
    while n != 1:
        sequence.append(n)
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
    sequence.append(1) # Append the final element '1' to the sequence
    return sequence

```

Ln 720, Col 1 Spaces: 4 UTF-8 CRLF [] Python 3.12.3 Python 3

```

File Edit Selection View Go ... ← → Q ALAC
ass_9_5.py Ass_10.2.py Ass_13.1.py X library.py library.html String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py Ass
Ass_13.1.py > generate_collatz_sequence
720 def generate_collatz_sequence(n):
    """
    if n <= 0:
        raise ValueError("Input must be a positive integer.")

    sequence = []
    while n != 1:
        sequence.append(n)
        if n % 2 == 0:
            n = n // 2
        else:
            n = 3 * n + 1
    sequence.append(1) # Append the final element '1' to the sequence
    return sequence
# Example usage
n = int(input("Enter a number: ")) # Get user input for the starting number
try:
    collatz_sequence = generate_collatz_sequence(n) # Generate the Collatz sequence
    print(collatz_sequence) # Print the generated sequence
except ValueError as e:
    print(e) # Print the error message if input is invalid

```

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.AC/
Enter a number: 155
[155, 466, 233, 700, 350, 175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167, 502, 251, 754, 377, 1132, 566, 283, 858, 425, 1276, 638, 319, 958, 479, 1438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 2734, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866, 433, 1308, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 168, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1]

PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>



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 dark-themed code editor interface. The central area displays Python code for generating the Lucas sequence. The code includes documentation comments (docstrings) explaining parameters, returns, examples, and error handling. The code uses a for loop to generate the sequence based on the input n.

```
769 #Requirement: Validate correctness with pytest.
770 def generate_Lucas_sequence(n):
    """
    Generate the Lucas number sequence up to n terms.

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

    Returns:
    list: A List containing the Lucas sequence up to n terms.

    Example:
    >>> generate_Lucas_sequence(5)
    [2, 1, 3, 4, 7]
    >>> generate_Lucas_sequence(1)
    [2]
    >>> generate_Lucas_sequence(-5)
    ValueError: Input must be a non-negative integer.
    >>> generate_Lucas_sequence(10)
    [2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")

    lucas_sequence = []
    for i in range(n):
        if i == 0:
            lucas_sequence.append(2)
        elif i == 1:
            lucas_sequence.append(1)
        else:
            next_lucas = lucas_sequence[i-1] + lucas_sequence[i-2]
            lucas_sequence.append(next_lucas)

    return lucas_sequence
```

The screenshot shows a light-themed code editor interface. The central area displays the same Python code for generating the Lucas sequence. The code structure and logic are identical to the one in the first screenshot. The code uses a for loop to generate the sequence based on the input n.

```
770 def generate_Lucas_sequence(n):
    """
    Generate the Lucas number sequence up to n terms.

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

    Returns:
    list: A List containing the Lucas sequence up to n terms.

    Example:
    >>> generate_Lucas_sequence(5)
    [2, 1, 3, 4, 7]
    >>> generate_Lucas_sequence(1)
    [2]
    >>> generate_Lucas_sequence(-5)
    ValueError: Input must be a non-negative integer.
    >>> generate_Lucas_sequence(10)
    [2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")

    lucas_sequence = []
    for i in range(n):
        if i == 0:
            lucas_sequence.append(2)
        elif i == 1:
            lucas_sequence.append(1)
        else:
            next_lucas = lucas_sequence[i-1] + lucas_sequence[i-2]
            lucas_sequence.append(next_lucas)

    return lucas_sequence
```



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

```
Ass_13.1.py > ...
820 def count_vowels_consonants(s):
821     """
822     Parameters:
823     s (str): The input string to be analyzed.
824
825     Returns:
826     tuple: A tuple containing the count of vowels and consonants in the format (vowel_count, consonant_count).
827
828     Example:
829     >>> count_vowels_consonants("hello")
830     (2, 3)
831     >>> count_vowels_consonants("")
832     (0, 0)
833     >>> count_vowels_consonants("aeiou")
834     (5, 0)
835     """
836     vowels = 'aeiouAEIOU'
837     vowel_count = sum(1 for char in s if char in vowels)
838     consonant_count = sum(1 for char in s if char.isalpha() and char not in vowels)
839
840     return vowel_count, consonant_count
841
842     # Example usage
843     input_string = input("Enter a string: ") # Get user input for the string
844
845
```

```
Ass_13.1.py > ...
816     #Large: Long text
817     #Only vowels: "aeiou" → (5,0)
818     #Requirement: Validate correctness with pytest.
819
820 def count_vowels_consonants(s):
821     """
822         Count the number of vowels and consonants in a given string.
823
824     Parameters:
825     s (str): The input string to be analyzed.
826
827     Returns:
828     tuple: A tuple containing the count of vowels and consonants in the format (vowel_count, consonant_count).
829
830     Example:
831     >>> count_vowels_consonants("hello")
832     (2, 3)
833     >>> count_vowels_consonants("")
834     (0, 0)
835     >>> count_vowels_consonants("aeiou")
836     (5, 0)
837     """
838     vowels = 'aeiouAEIOU'
839     vowel_count = sum(1 for char in s if char in vowels)
840
841     print("Enter a string: Hruthika")
842     print("Vowels: 3, Consonants: 5")
843
```