

# 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

```
#2303A51543
#Task Description #1 (Refactoring – Removing Code Duplication)
# 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))
#Prompt:Identify duplicate logic and replace it with functions or classes.o Add docstring
s to all functions.
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.

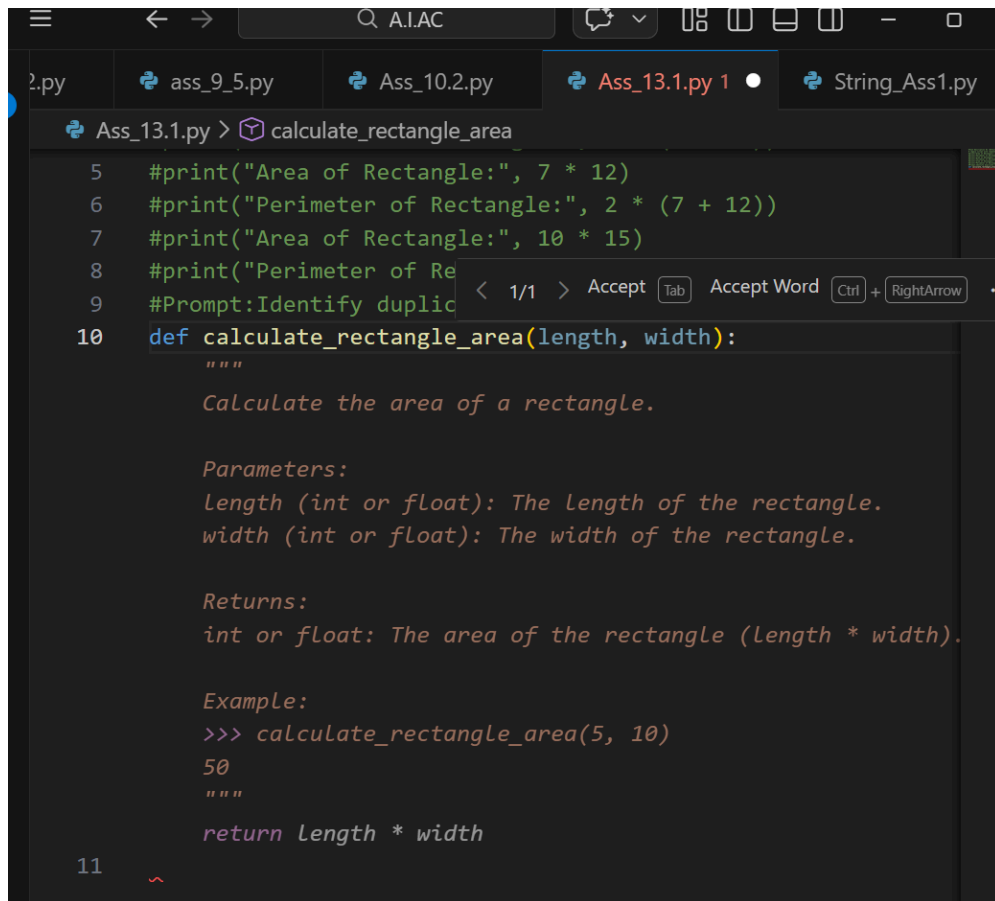
    Returns:
    int or float: The area of the rectangle (length * width).

    Example:
    >>> calculate_rectangle_area(5, 10)
```

```

50
"""
    return length * width
def calculate_rectangle_perimeter(length, width):
    """
    Calculate the perimeter 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 perimeter of the rectangle (2 * (length + width)).
    Example:
    >>> calculate_rectangle_perimeter(5, 10)
    30
    """
    return 2 * (length + width)
# Example usage
lengths = [5, 7, 10]
widths = [10, 12, 15]
for length, width in zip(lengths, widths):
    area = calculate_rectangle_area(length, width)
    perimeter = calculate_rectangle_perimeter(length, width)
    print(f"Area of Rectangle: {area}")
    print(f"Perimeter of Rectangle: {perimeter}")

```



The image shows a code editor window with several tabs at the top: '2.py', 'ass\_9\_5.py', 'Ass\_10.2.py', 'Ass\_13.1.py 1', and 'String\_Ass1.py'. The active tab is 'Ass\_13.1.py', and the cursor is positioned at line 10, column 1, where a function definition is being entered. The function is named 'calculate\_rectangle\_area' and takes two parameters: 'length' and 'width'. The function body is currently empty, but it includes a docstring with the following 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 Rectangle:", 2 * (10 + 15))
9 #Prompt:Identify duplicate
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
```

```

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)

```

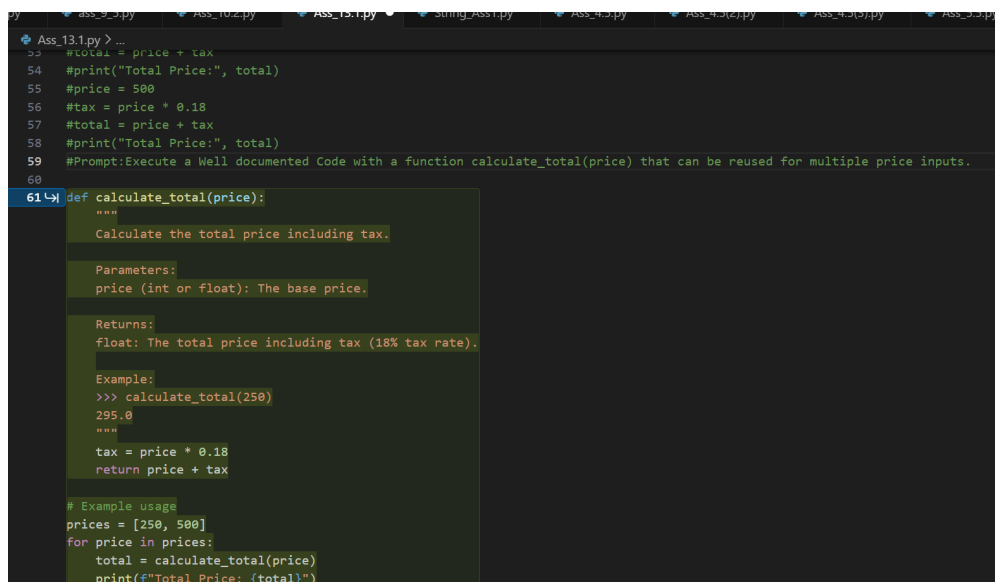
- Expected Output:
  - Code with a function calculate\_total(price) that can be reused for multiple price inputs.
  - Well documented code

```

#Task Description #2 (Refactoring – Extracting Reusable Functions)
# 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)
#Prompt:Execute a Well documented Code with a function calculate_total(price) that can be
reused for multiple price inputs.
def calculate_total(price):
    """
    Calculate the total price including tax.
    Parameters:
    price (int or float): The original price of the item.
    Returns:
    int or float: The total price after adding 18% tax.
    Example:
    >>> calculate_total(250)
    295.0
    """
    tax = price * 0.18 # Calculate tax as 18% of the price
    total = price + tax # Add tax to the original price to get the total
    return total

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

```



```

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.

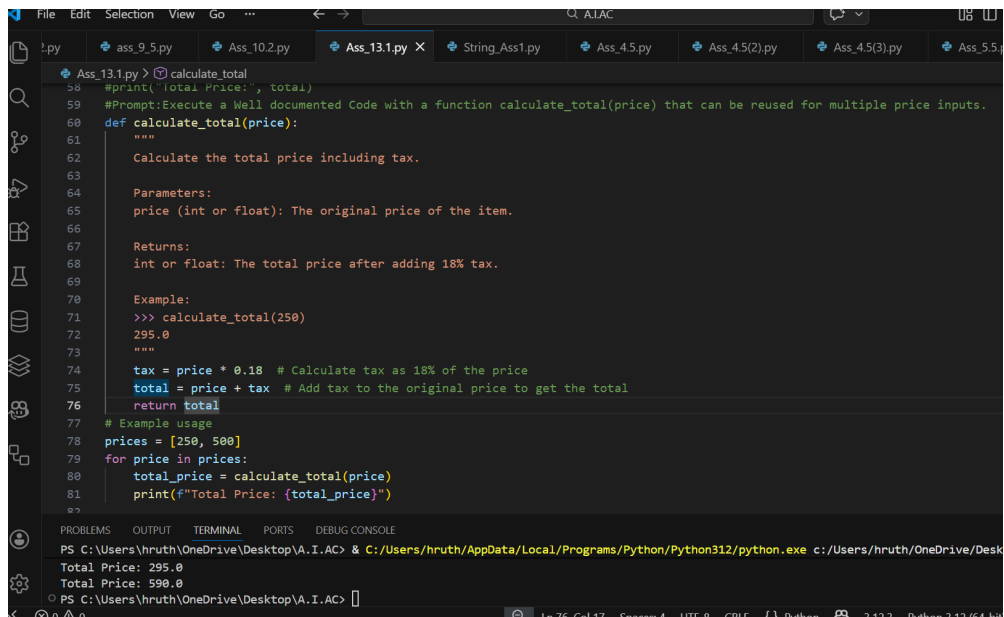
    Parameters:
    price (int or float): The base price.

    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}")

```



The screenshot shows a Python IDE with a dark theme. The top toolbar includes File, Edit, Selection, View, Go, and a search icon. The tab bar shows several files: ass\_9.5.py, Ass\_10.2.py, Ass\_13.1.py (active), String\_Ass1.py, Ass\_4.5.py, Ass\_4.5(2).py, Ass\_4.5(3).py, and Ass\_5.5.py. The editor displays the following code:

```
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
```

The bottom panel shows the OUTPUT tab with the following text:

```
PS C:\Users\hruth\OneDrive\Desktop\VA.I.AC> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desk
Total Price: 295.0
Total Price: 590.0
PS C:\Users\hruth\OneDrive\Desktop\VA.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  $\geq 90$  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.

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.

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

#Prompt: 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  $\geq 90$  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.

#Include proper docstrings for:

#The class

#The method (with parameter and return descriptions)

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

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"
    else:
        raise ValueError("Marks should be between 0 and 100.")

# Example usage
grade_calculator = GradeCalculator()
marks_list = [85, 72, 95, 65, 30]
for marks in marks_list:
    grade = grade_calculator.calculate_grade(marks)
    print(f"Marks: {marks}, Grade: {grade}")

```

```
File Edit Selection View ... AIAC
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).p
Ass_13.1.py > ...
114 # The class
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"
        else:
            raise ValueError("Marks should be between 0 and 100.")

# Example usage
grade_calculator = GradeCalculator()
marks_list = [85, 72, 95, 65, 30]
for marks in marks_list:
    grade = grade_calculator.calculate_grade(marks)
    print(f"Marks: {marks}, Grade: {grade}")
```

```
File Edit Selection View ... AIAC
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
Ass_13.1.py > GradeCalculator > calculate_grade
117 class GradeCalculator:
124 def calculate_grade(self, marks):
134     """
135     Grading Logic:
136     Marks ≥ 90 and ≤ 100 → "Grade A"
137     Marks ≥ 80 → "Grade B"
138     Marks ≥ 70 → "Grade C"
139     Marks ≥ 40 → "Grade D"
140     Marks ≥ 0 → "Fail"
141     """
142     if 90 <= marks <= 100:
143         return "Grade A"
144     elif marks >= 80:
145         return "Grade B"
146     elif marks >= 70:
147         return "Grade C"
148     elif marks >= 40:
149         return "Grade D"
150     elif marks >= 0:
151         return "Fail"
152     else:
153         raise ValueError("Marks should be between 0 and 100.")
154 # Example usage
155 grade_calculator = GradeCalculator()
156 marks_list = [85, 72, 95, 65, 30]
157 for marks in marks_list:
158     grade = grade_calculator.calculate_grade(marks)
159     print(f"Marks: {marks}, Grade: {grade}")
160
```

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



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

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

Instructions:

- o Identify input, processing, and output sections.
- o Convert each into a separate function.
- o Improve code readability without changing behavior.

- Sample Legacy Code:

```
num = int(input("Enter number: "))
square = num * num
print("Square:", square)
```

- Expected Output:

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

```
#Task Description #4 (Refactoring – Converting Procedural Code to Functions)
#Sample Legacy Code:
#num = int(input("Enter number: "))
#square = num * num
#print("Square:", square)
#Prompt:Generate a modular code with docstrings using functions like get_input(), calculate_square(), and display_result().To refactor procedural input–processing logic into functions.
def get_input():
    """
    Prompt the user to enter a number and return it as an integer.
    Returns:
    int: The number entered by the user.
    """
    return int(input("Enter number: "))
def calculate_square(num):
    """
    Calculate the square of a given number.
    Parameters:
    num (int): The number to be squared.
    Returns:
    int: The square of the input number.
    """
    return num * num
def display_result(square):
    """
    Display the result of the square calculation.
    Parameters:
    square (int): The squared value to be displayed.
    """
    print("Square:", square)
# Main function to orchestrate the flow of the program
def main():
    number = get_input() # Get user input
```

```

    square = calculate_square(number) # Calculate the square of the input number
    display_result(square) # Display the result
# Execute the main function
if __name__ == "__main__":
    main()

```

```

9
10 #Task Description #4 (Refactoring - Converting Procedural Code to Functions)
11 #Sample Legacy Code:
12 #num = int(input("Enter number: "))
13 #square = num * num
14 #print("Square:", square)
15 #Prompt:Generate a modular code with docstrings using functions like get_input(), calculate_square(), and display_result()
16 def get_input():
17     """
18     Prompt the user to enter a number and return it as an integer.
19
20     Returns:
21     int: The number entered by the user.
22     """
23     return int(input("Enter number: "))
24
25
26
27
28
29

```

```

File Edit Selection View ... ← → AIAC
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.

```



### 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.

#Task 5 (Refactoring Procedural Code into OOP Design)

#Legacy Code:

#salary = 50000

#tax = salary \* 0.2

#net = salary - tax

#print(net)

#Prompt: Refactor procedural code into a class-based design like EmployeeSalaryCalculator with methods and attributes.

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

y

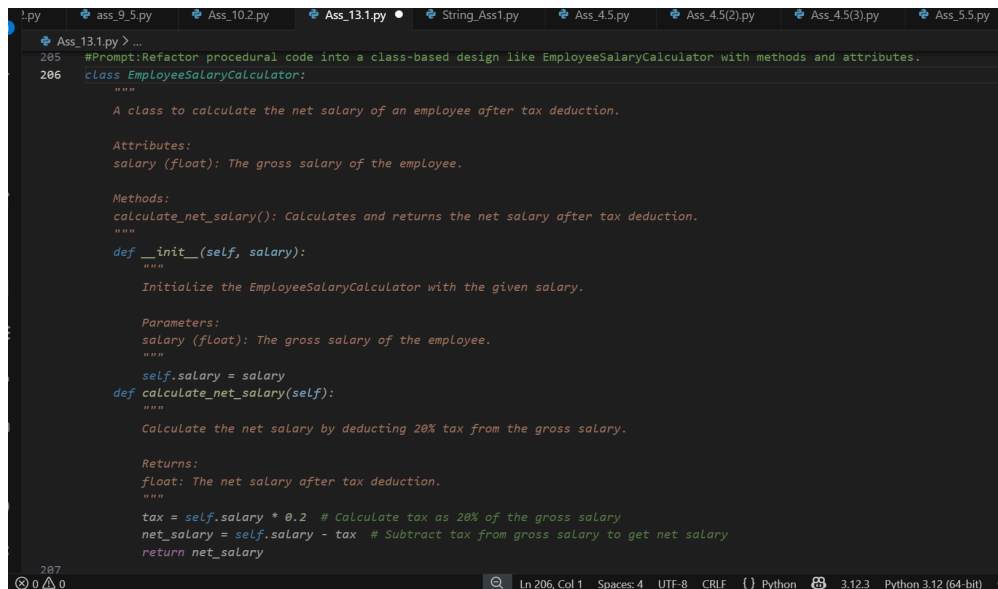
return net\_salary

# Example usage

employee\_salary\_calculator = EmployeeSalaryCalculator(50000) # Create an instance with a salary of 50000

net\_salary = employee\_salary\_calculator.calculate\_net\_salary() # Calculate the net salary

y  
print(net\_salary) # Expected output: 40000.0



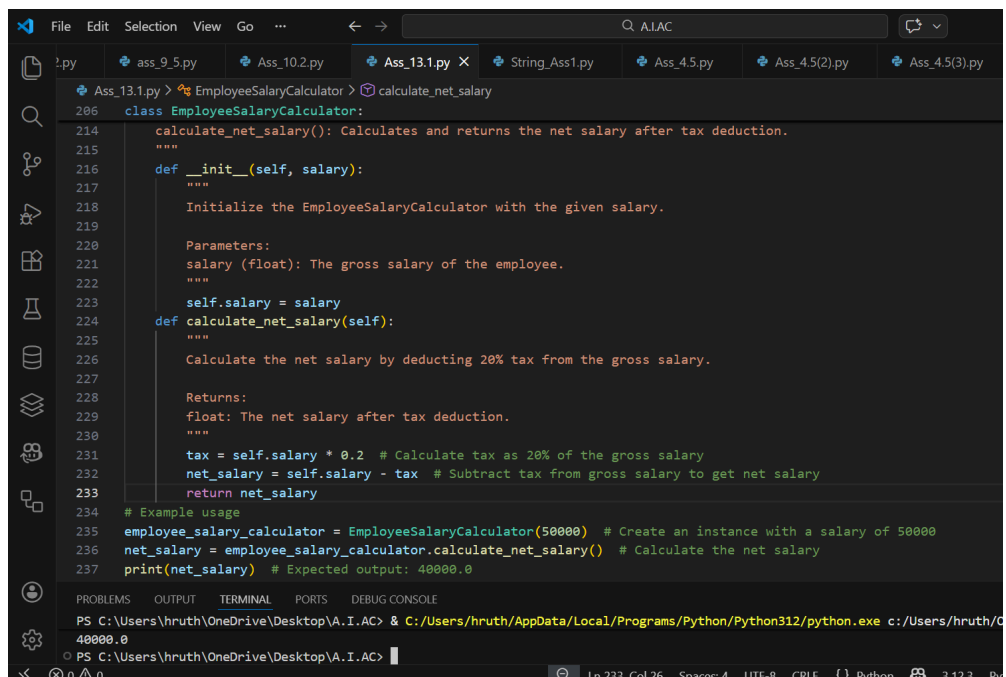
```
!py  ass_9_5.py  Ass_10_2.py  Ass_13.1.py  String_Ass1.py  Ass_4.5.py  Ass_4.5(2).py  Ass_4.5(3).py  Ass_5.5.py
Ass_13.1.py > ...
205 #Prompt:Refactor procedural code into a class-based design like EmployeeSalaryCalculator with methods and attributes.
206 class EmployeeSalaryCalculator:
    """
    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
Ln 206, Col 1  Spaces: 4  UTF-8  CRLF  Python  3.12.3  Python 3.12 (64-bit)
```



```
File  Edit  Selection  View  Go  ...  AIAC
!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_13.1.py > EmployeeSalaryCalculator > calculate_net_salary
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
239
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
40000.0
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
Ln 233, Col 26  Spaces: 4  UTF-8  CRLF  Python  3.12.3  Py
```



### Task 6 (Optimizing Search Logic)

- **Task: Refactor inefficient linear searches using appropriate data structures.** Focus Areas:
  - o Time complexity
  - o Data structure choice

Legacy Code:

```
users = ["admin", "guest", "editor", "viewer"]
name = input("Enter username: ")
found = False
for u in users:
    if u == name:
        found = True
print("Access Granted" if found else "Access Denied")
```

#### Expected Outcome:

- o Use of sets or dictionaries with complexity justification

```
#Task 6 (Optimizing Search Logic)
#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")
#Prompt: Refactor inefficient linear searches using appropriate data structures.
#Output: Use of sets or dictionaries with complexity justification
#Focus Areas: Time complexity and Data structure choice
def check_user_access(username, user_set):
    """
    Check if the given username exists in the user set and return access status.
    Parameters:
    username (str): The username to be checked.
    user_set (set): A set of valid usernames for access control.
    Returns:
    str: "Access Granted" if the username is found, otherwise "Access Denied".
    Time Complexity:
    The use of a set allows for O(1) average time complexity for membership checks,
    making it more efficient than a list which has O(n) time complexity for the same operation.
    """
    return "Access Granted" if username in user_set else "Access Denied"
# Example usage
users = {"admin", "guest", "editor", "viewer"} # Using a set
name = input("Enter username: ") # Get username input from the user
access_status = check_user_access(name, users) # Check access status
print(access_status) # Print the access status
```

```
File Edit Selection View Go ...
Ass_13.1.py
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):
    """
    Check if the given username exists in the user set and return access status.

    Parameters:
    username (str): The username to be checked.
    user_set (set): A set of valid usernames for access control.

    Returns:
    str: "Access Granted" if the username is found, otherwise "Access Denied".

    Time Complexity:
    The use of a set allows for O(1) average time complexity for membership checks,
    making it more efficient than a list which has O(n) time complexity for the same operation.
    """
    return "Access Granted" if username in user_set else "Access Denied"
```

```
File Edit Selection View Go ...
Ass_13.1.py X
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"
267 # Example usage
268 users = {"admin", "guest", "editor", "viewer"} # Using a set
269 name = input("Enter username: ") # Get username input from the user
270 access_status = check_user_access(name, users) # Check access status
271 print(access_status) # Print the access status

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.")

```

```

#Task 7 – Refactoring the Library Management System
# 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.")
#Prompt: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.
# library.py
class Library:
    """
    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}
            return "Book added successfully."
        else:
            return "Book already exists."
    def 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.
    """
    if isbn in self.library_db:
        del self.library_db[isbn]
        return "Book removed successfully."
    else:
        return "Book not found."
def 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.
    """
    if isbn in self.library_db:
        return f"Book Found: {self.library_db[isbn]}"
    else:
        return "Book not found."
# Example usage
library = Library() # Create an instance of the Library class
print(library.add_book("Python Basics", "John Doe", "101")) # Add first book
print(library.add_book("AI Fundamentals", "Jane Smith", "102")) # Add second book
print(library.search_book("101")) # Search for the first book
print(library.remove_book("101")) # Remove the first book
print(library.search_book("101")) # Search for the first book again after removal

```

```
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}
            return "Book added successfully."
        else:
            return "Book already exists."

    def 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.
        """
        if isbn in self.library_db:
            del self.library_db[isbn]
            return "Book removed successfully."
        else:
            return "Book not found."

    def 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.
        """
        if isbn in self.library_db:
            book_info = self.library_db[isbn]
            return f"Book Found: {'title': '{book_info['title']}', 'author': '{book_info['author']}'}"
        else:
            return "Book not found."

# Example usage
library = Library()
print(library.add_book("Python Basics", "John Doe", "101")) # Add first book
print(library.add_book("AI Fundamentals", "Jane Smith", "102")) # Add second book
print(library.search_book("101")) # Search for the first book
print(library.remove_book("101")) # Remove the first book
print(library.search_book("101")) # Search for the first book again after removal
```

```
File Edit Selection View Go ... A.I.A.C
Ass_13.1.py X String_Ass1.py Ass_4.5.py Ass_4.5(2).py Ass_4.5(3).py
Ass_13.1.py > Library > search_book
324 class Library:
371 def search_book(self, isbn):
373     Search for a book in the library database using its ISBN.
374
375     Parameters:
376         isbn (str): The ISBN number of the book to be searched.
377
378     Returns:
379         str: A message indicating whether the book was found along with its details or if it was not found.
380     """
381     if isbn in self.library_db:
382         return f"Book Found: {self.library_db[isbn]}"
383     else:
384         return "Book not found."
385
386 # Example usage
387 library = Library() # Create an instance of the Library class
388 print(library.add_book("Python Basics", "John Doe", "101")) # Add first book
389 print(library.add_book("AI Fundamentals", "Jane Smith", "102")) # Add second book
390 print(library.search_book("101")) # Search for the first book
391 print(library.remove_book("101")) # Remove the first book
392 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.A.C> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.exe c:/Users/hruth/OneDrive/Desktop/A.I.A.C/Ass_13.1.py
Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
PS C:\Users\hruth\OneDrive\Desktop\A.I.A.C>
```

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> python -m pydoc library
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.

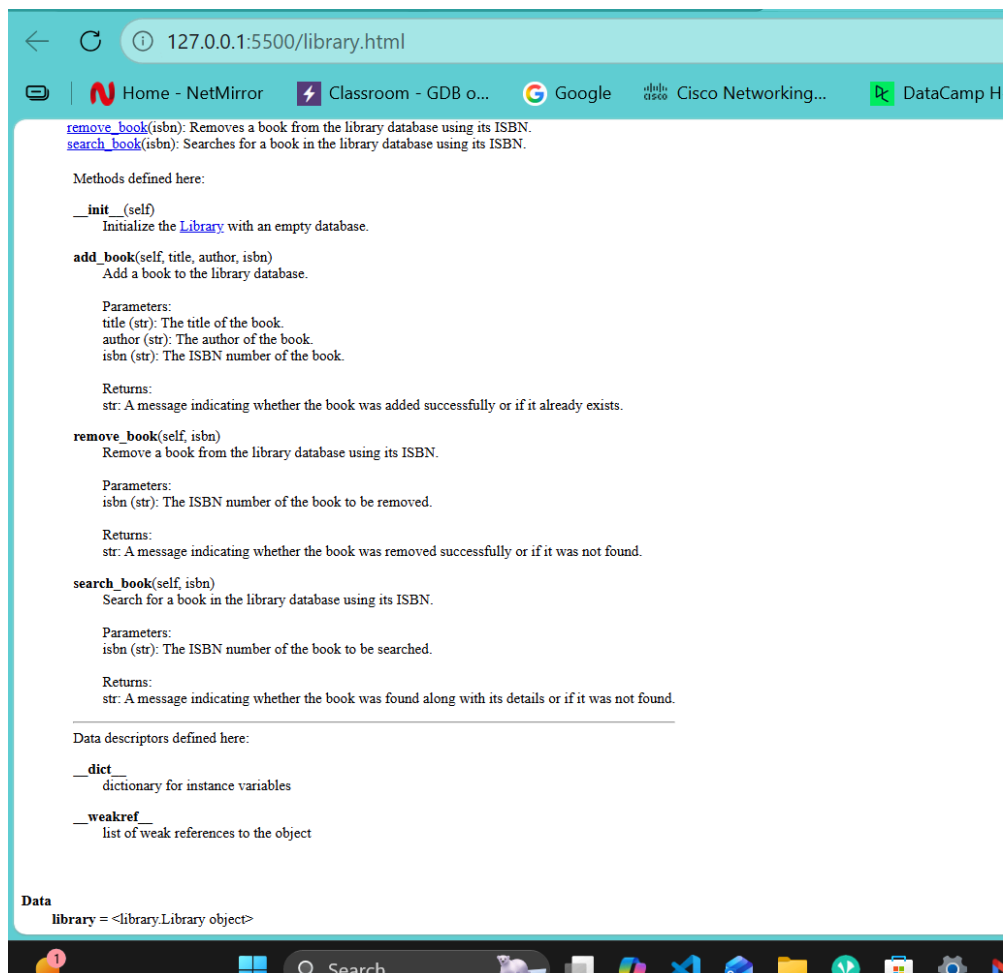
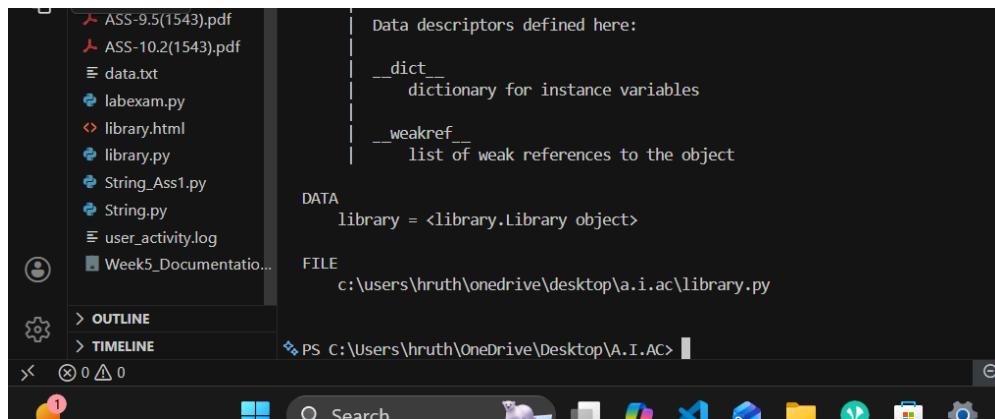
Returns:
  str: A message indicating whether the book was removed successfully or if it was not found.

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_
```



**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
```

```
#Task 8– Fibonacci Generator
#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
#Prompt:Refactor into a clean reusable function (generate_fibonacci).Add docstrings and test cases.Compare AI-refactored vs original.
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
# Example usage
print(generate_fibonacci(10)) # Expected output: [0, 1,1, 2, 3, 5, 8, 13, 21, 34]

```

```

Ass_13.1.py > ...
480 #print(b)
481 #for i in range(2,n):
482 # c=a+b
483 # print(c)
484 # a=b
485 # b=c
486 #Prompt:Refactor into a clean reusable function (generate_fibonacci).Add docstrings and test cases.Compare AI-refactored vs original.
487 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
488
489

```

```
Ass_13.1.py > ...
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):
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\hruth\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")
```

```

#Task 9 – Twin Primes Checker
#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")
#Prompt:Refactor into is_prime(n) and is_twin_prime(p1, p2).Add docstrings and optimize.G
enerate a list of twin primes in a given range.
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
def is_twin_prime(p1, p2):
    """
    Check if two numbers are twin primes.
    Parameters:
    p1 (int): The first prime number.
    p2 (int): The second prime number.
    Returns:
    bool: True if the numbers are twin primes, False otherwise.
    Example:
    >>> is_twin_prime(11, 13)
    True
    >>> is_twin_prime(12, 14)

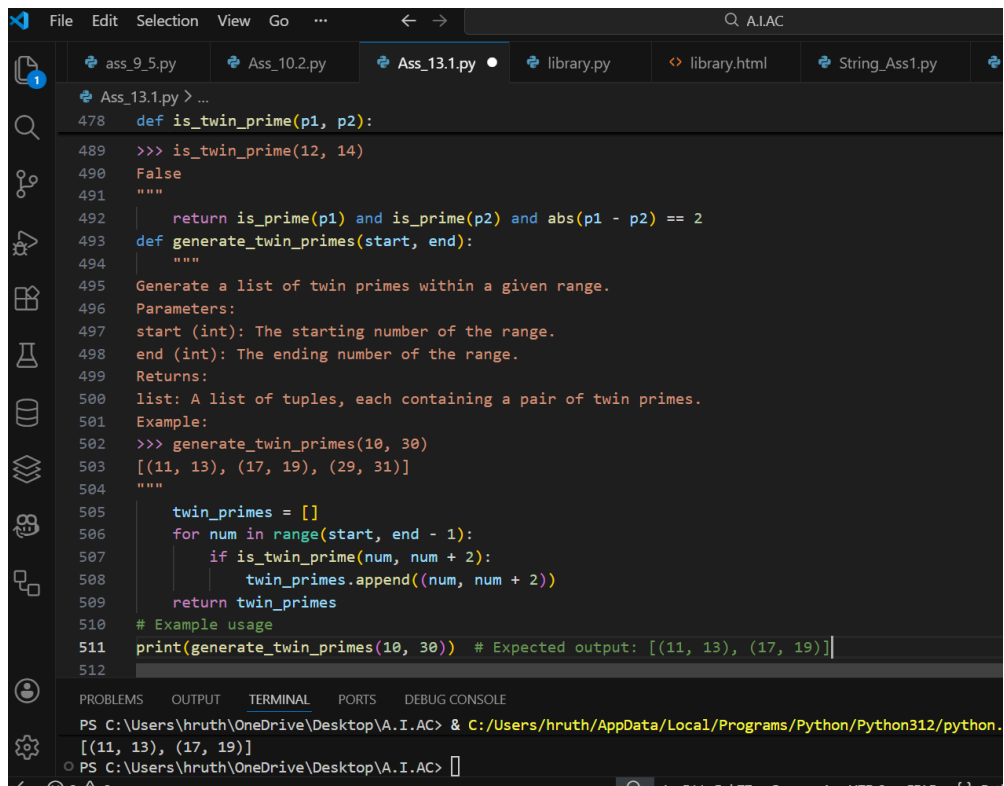
```

```

False
"""
    return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
def generate_twin_primes(start, end):
    """
    Generate a list of twin primes within a given range.
    Parameters:
    start (int): The starting number of the range.
    end (int): The ending number of the range.
    Returns:
    list: A list of tuples, each containing a pair of twin primes.
    Example:
    >>> generate_twin_primes(10, 30)
    [(11, 13), (17, 19), (29, 31)]
    """
    twin_primes = []
    for num in range(start, end - 1):
        if is_twin_prime(num, num + 2):
            twin_primes.append((num, num + 2))
    return twin_primes
# Example usage
print(generate_twin_primes(10, 30)) # Expected output: [(11, 13), (17, 19)]

```

The screenshot shows a code editor with a dark theme. The active file is 'Ass\_13.1.py'. The code defines a function `is_prime(n)` with a docstring that includes a prompt to refactor into `is_prime(n)` and `is_twin_prime(p1, p2)`, and to add docstrings and optimize. The function checks if a number is prime by testing divisibility from 2 to  $\sqrt{n}$ . The editor shows lines 455 and 456. Line 455 is a comment: `#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.`. Line 456 is the function definition: `def is_prime(n):`. The docstring includes: '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.', and an 'Example:' section with test cases: `>>> is_prime(11)` returns `True`, and `>>> is_prime(12)` returns `False`. The function body starts with `if n <= 1: return False`, followed by a loop `for i in range(2, int(n**0.5) + 1):` with an inner `if n % i == 0: return False`, and finally `return True`. The line number 457 is visible at the bottom left of the editor window.



The image shows a code editor with a dark theme. The top bar includes a search icon and the text 'A.I.A.C'. Below the top bar, there are tabs for 'ass\_9\_5.py', 'Ass\_10.2.py', 'Ass\_13.1.py' (which is active), 'library.py', 'library.html', and 'String\_Ass1.py'. The main editor area displays Python code for finding twin primes. The code includes a function `is_twin_prime` and a function `generate_twin_primes`. The `generate_twin_primes` function has a docstring explaining its purpose and parameters. Below the code, there is a terminal window with the following text:

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\Users\hruth\OneDrive\Desktop\A.I.A.C> & C:/Users/hruth/AppData/Local/Programs/Python/Python312/python.
[(11, 13), (17, 19)]
PS C:\Users\hruth\OneDrive\Desktop\A.I.A.C> |
```



## Task 10 – Refactoring the Chinese Zodiac Program

### Objective

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

The current program reads a year from the user and prints the corresponding Chinese Zodiac sign.

However, the implementation contains repetitive conditional logic, lacks modular design, and does not follow clean coding principles.

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

### Chinese Zodiac Cycle (Repeats Every 12 Years)

1. Rat
2. Ox
3. Tiger
4. Rabbit
5. Dragon
6. Snake
7. Horse
8. Goat (Sheep)
9. Monkey
10. Rooster
11. Dog
12. Pig

### # Chinese Zodiac Program (Unstructured Version)

#### # This code needs refactoring.

```
year = int(input("Enter a year: "))
if year % 12 == 0:
    print("Monkey")
elif year % 12 == 1:
    print("Rooster")
elif year % 12 == 2:
    print("Dog")
elif year % 12 == 3:
    print("Pig")
elif year % 12 == 4:
    print("Rat")
elif year % 12 == 5:
    print("Ox")
elif year % 12 == 6:
    print("Tiger")
elif year % 12 == 7:
    print("Rabbit")
elif year % 12 == 8:
    print("Dragon")
elif year % 12 == 9:
    print("Snake")
elif year % 12 == 10:
    print("Horse")
```

```
elif year % 12 == 11:
    print("Goat")
```

**You must:**

1. Create a reusable function: `get_zodiac(year)`
2. Replace the if-elif chain with a cleaner structure (e.g., list or dictionary).
3. Add proper docstrings.
4. Separate input handling from logic.
5. Improve readability and maintainability.
6. Ensure output remains correct.

**#Task 10 – Refactoring the Chinese Zodiac Program**

**# 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")
```

**#Prompt: Create a reusable function: `get_zodiac(year)`**

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

**#Add proper docstrings.**

**#Separate input handling from logic.**

**#Improve readability and maintainability.**

**#Ensure output remains correct.**

```
def get_zodiac(year):
```

```
    """
```

```
    Get the Chinese Zodiac sign for a given year.
```

```
    Parameters:
```

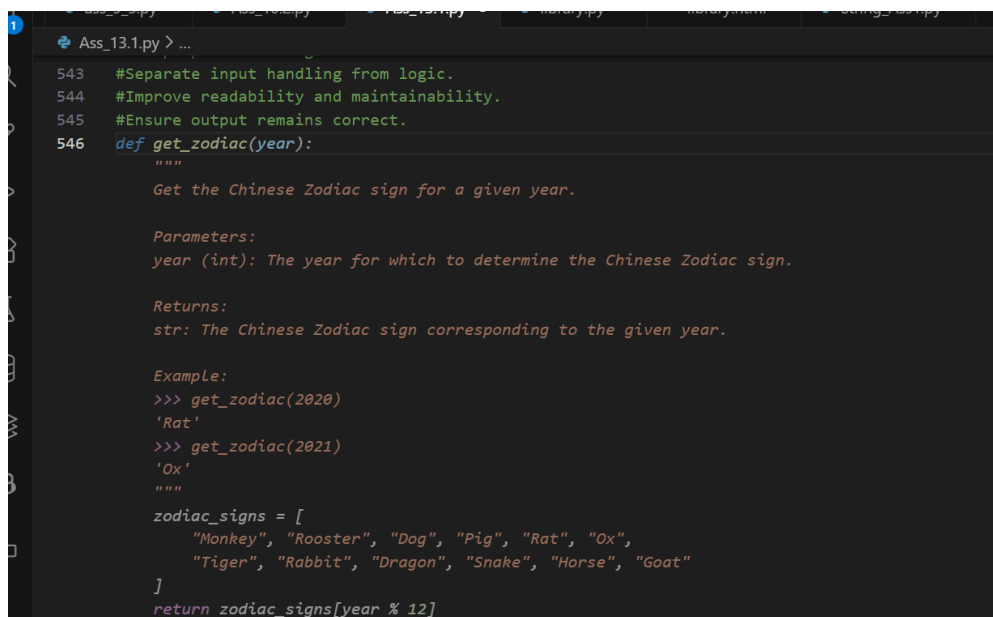
```

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]

# Example usage
year = int(input("Enter a year: ")) # Get user input for the year
zodiac_sign = get_zodiac(year) # Get the corresponding Chinese Zodiac sign
print(zodiac_sign) # Print the Zodiac sign

```



```

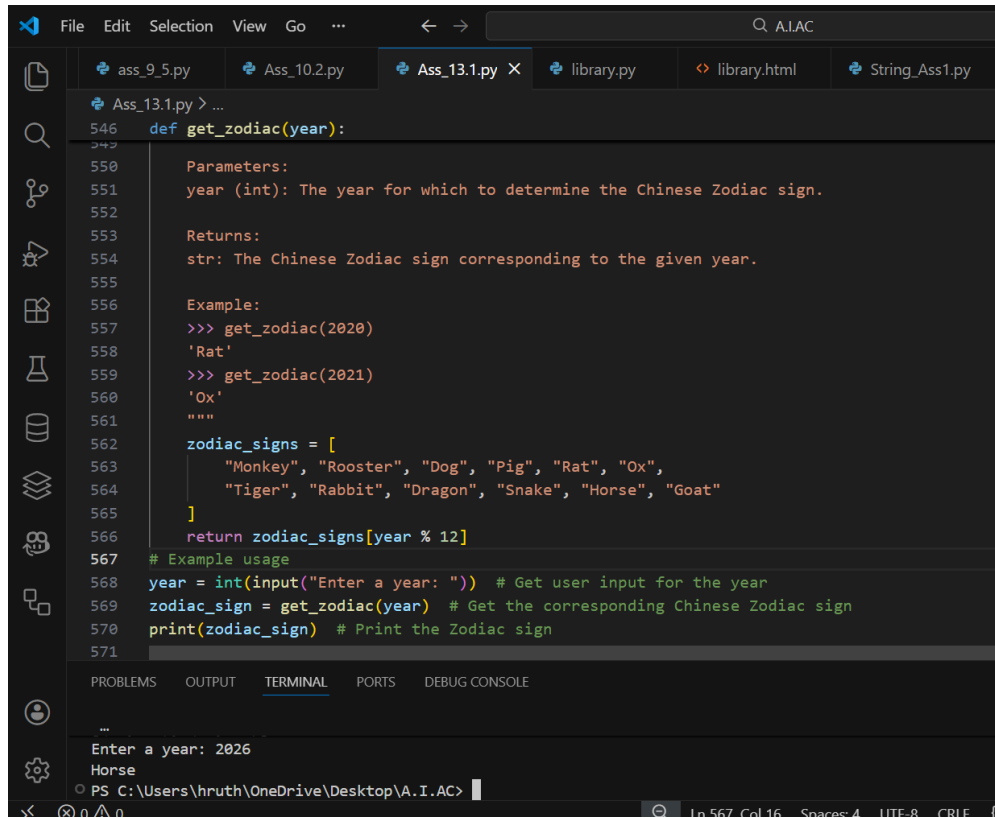
Ass_13.1.py
543 #Separate input handling from logic.
544 #Improve readability and maintainability.
545 #Ensure output remains correct.
546 def get_zodiac(year):
    """
    Get the Chinese Zodiac sign for a given year.

    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 with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, and a search bar with the text 'A.I.AC'. The file explorer on the left shows several files: ass\_9\_5.py, Ass\_10.2.py, Ass\_13.1.py (selected), library.py, library.html, and String\_Ass1.py. The main editor area displays the code for 'Ass\_13.1.py'. The code defines a function 'get\_zodiac(year)' with a docstring that includes parameters, returns, and an example. The function uses a list 'zodiac\_signs' to map years to zodiac signs. Below the function, there is an example usage section that takes user input for a year and prints the corresponding zodiac sign. The bottom panel shows the terminal output where the user has entered '2026' and the program has output 'Horse'.

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

PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE

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



### 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:**

- $18 \rightarrow 1 + 8 = 9 \rightarrow 18 \div 9 = 2$  ✓ (Harshad Number)
- $19 \rightarrow 1 + 9 = 10 \rightarrow 19 \div 10 \neq \text{integer}$  ✗ (Not Harshad)

#### Problem Statement

The current implementation:

- Mixes logic and input handling
- Uses redundant variables
- Does not use reusable functions properly
- Returns print statements instead of boolean values
- Lacks documentation

You must refactor the code to follow clean coding principles.

# Harshad Number Checker (Unstructured Version)

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

```
temp = num
```

```
sum_digits = 0
```

```
while temp > 0:
```

```
    digit = temp % 10
```

```
    sum_digits = sum_digits + digit
```

```
    temp = temp // 10
```

```
if sum_digits != 0:
```

```
    if num % sum_digits == 0:
```

```
        print("True")
```

```
    else:
```

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

```
else:
```

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

**You must:**

1. Create a reusable function: `is_harshad(number)`
2. The function must:
  - 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).

```
#Task 11 – Refactoring the Harshad (Niven) Number Checker
```

```
#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")
#Prompt: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. Ensure the program handles edge cases (e.g., 0, negative numbers).
def is_harshad(number):
    """
    Check if a number is a Harshad (Niven) number.
    Parameters:
    number (int): The number to be checked.
    Returns:
    bool: True if the number is a Harshad number, False otherwise.
    A Harshad number is an integer that is divisible by the sum of its digits.
    Example:
    >>> is_harshad(18)
    True
    >>> is_harshad(19)
    False
    >>> is_harshad(0)
    False
    >>> is_harshad(-12)
    False
    """
    if number < 0:
        return False # Harshad numbers are typically defined for non-negative integers
    temp = abs(number) # Use absolute value to handle negative numbers
    sum_digits = 0
    while temp > 0:
        digit = temp % 10
        sum_digits += digit
        temp //= 10
    if sum_digits == 0:
        return False # Avoid division by zero
    return number % sum_digits == 0
# Example usage
num = int(input("Enter a number: ")) # Get user input for the number

```

```
result = is_harshad(num) # Check if the number is a Harsh
print(result) # Print the result (True or False)
```

```
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     if number == 0:
613         return False
614     sum_digits = sum(int(digit) for digit in str(abs(number)))
615     if sum_digits == 0:
616         return False
617     return number % sum_digits == 0
618
619 # Example usage
620 number = int(input("Enter a number: "))
621 print(is_harshad(number))
```

The image shows a Visual Studio Code editor window with a dark theme. The top menu bar includes File, Edit, Selection, View, Go, and a search bar containing 'A.I.A.C.'. The file explorer on the left shows several files: 'ass\_9\_5.py', 'Ass\_10.2.py', 'Ass\_13.1.py' (selected), 'library.py', 'library.html', 'String\_Ass1.py', and 'Ass\_4.5.p'. The main editor area displays the code for 'Ass\_13.1.py', which defines a function 'is\_harshad' and includes example usage. The code is as follows:

```
596 def is_harshad(number):
611     >>> is_harshad(19)
612     False
613     >>> is_harshad(0)
614     False
615     >>> is_harshad(-12)
616     False
617     """
618     if number < 0:
619         return False # Harshad numbers are typically defined for non-negative integers
620     temp = abs(number) # Use absolute value to handle negative numbers
621     sum_digits = 0
622     while temp > 0:
623         digit = temp % 10
624         sum_digits += digit
625         temp //= 10
626     if sum_digits == 0:
627         return False # Avoid division by zero
628     return number % sum_digits == 0
629 # Example usage
630 num = int(input("Enter a number: ")) # Get user input for the number
631 result = is_harshad(num) # Check if the number is a Harsh
632 print(result) # Print the result (True or False)
633
```

Below the code editor, the 'TERMINAL' tab is active, showing the execution of the script. It displays the prompt 'Enter a number: 18' followed by the output 'True'. The terminal path is 'PS C:\Users\hruth\OneDrive\Desktop\A.I.A.C>'.



### 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).

```
#Task 12 – Refactoring the Factorial Trailing Zeros Program
#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)
#Prompt: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).
def count_trailing_zeros(n):
    """
    Count the number of trailing zeros in the factorial of a given non-negative integer.
    Parameters:
    n (int): A non-negative integer for which to count trailing zeros in n!.
    Returns:
    int: The number of trailing zeros in n!.
    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 fact
    orial.
    Example:
    >>> count_trailing_zeros(100)
    24
    >>> count_trailing_zeros(0)
    0
    >>> count_trailing_zeros(-5)
    ValueError: Input must be a non-negative integer.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")
    count = 0
    power_of_5 = 5
    while power_of_5 <= n:
        count += n // power_of_5
        power_of_5 *= 5
    return count
# Example usage
n = int(input("Enter a number: ")) # Get user input for the number
try:
    trailing_zeros = count_trailing_zeros(n) # Count the trailing zeros in n!
    print("Trailing zeros:", trailing_zeros) # Print the result
except ValueError as e:
    print(e) # Print the error message if input is invalid

```

```
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_13.1.py > ...
655 def count_trailing_zeros(n):
    """
    Count the number of trailing zeros in the factorial of a given non-negative integer.

    Parameters:
    n (int): A non-negative integer for which to count trailing zeros in n!.

    Returns:
    int: The number of trailing zeros in n!.

    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.

    Example:
    >>> count_trailing_zeros(100)
    24
    >>> count_trailing_zeros(0)
    0
    >>> count_trailing_zeros(-5)
    ValueError: Input must be a non-negative integer.
    """
    if n < 0:
        raise ValueError("Input must be a non-negative integer.")

    count = 0
    power_of_5 = 5
    while power_of_5 <= n:
        count += n // power_of_5
        power_of_5 *= 5
```

```
Ass_9.5.py | Ass_10.2.py | Ass_13.1.py X | library.py | library.html | String_Ass1.py | Ass_13.1.py > ...
655 def count_trailing_zeros(n):
672     0
673     >>> count_trailing_zeros(-5)
674     ValueError: Input must be a non-negative integer.
675     """
676     if n < 0:
677         raise ValueError("Input must be a non-negative integer.")
678
679     count = 0
680     power_of_5 = 5
681     while power_of_5 <= n:
682         count += n // power_of_5
683         power_of_5 *= 5
684
685     return count
686 # Example usage
687 n = int(input("Enter a number: ")) # Get user input for the number
688 try:
689     trailing_zeros = count_trailing_zeros(n) # Count the trailing zeros in n!
690     print("Trailing zeros:", trailing_zeros) # Print the result
691 except ValueError as e:
692     print(e) # Print the error message if input is invalid
693
```

PROBLEMS | OUTPUT | TERMINAL | PORTS | DEBUG CONSOLE

```
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC> ...
Enter a number: 15430900
Trailing zeros: 3857720
PS C:\Users\hruth\OneDrive\Desktop\A.I.AC>
```



## Test Cases Design

### 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:
  - If  $n$  is even  $\rightarrow$  next =  $n / 2$ .
  - 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  $\rightarrow 6/2 = 3$ )
- 3 (odd  $\rightarrow 3*3+1 = 10$ )
- 10 (even  $\rightarrow 10/2 = 5$ )
- 5 (odd  $\rightarrow 3*5+1 = 16$ )
- 16 (even  $\rightarrow 16/2 = 8$ )
- 8 (even  $\rightarrow 8/2 = 4$ )
- 4 (even  $\rightarrow 4/2 = 2$ )
- 2 (even  $\rightarrow 2/2 = 1$ )

Output:

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

```
#Task 13 (Collatz Sequence Generator – Test Case Design)
#Prompt:Generate Collatz sequence until reaching 1.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  $\rightarrow$  next =  $n / 2$ .
#If n is odd  $\rightarrow$  next =  $3n + 1$ .
#Repeat until we reach 1.
#Return the full sequence as a list.
#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.
#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 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

# 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

```

```

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:

```

```

Ass_13.1.py > generate_collatz_sequence
720 def generate_collatz_sequence(n):
    """
    741
    742     if n <= 0:
    743         raise ValueError("Input must be a positive integer.")
    744
    745     sequence = []
    746     while n != 1:
    747         sequence.append(n)
    748         if n % 2 == 0:
    749             n = n // 2
    750         else:
    751             n = 3 * n + 1
    752     sequence.append(1) # Append the final element '1' to the sequence
    753     return sequence
    754
    755 # Example usage
    756 n = int(input("Enter a number: ")) # Get user input for the starting number
    757 try:
    758     collatz_sequence = generate_collatz_sequence(n) # Generate the Collatz sequence
    759     print(collatz_sequence) # Print the generated sequence
    760 except ValueError as e:
    761     print(e) # Print the error message if input is invalid
    """

```

```

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, 850, 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, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 106, 53, 160, 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)

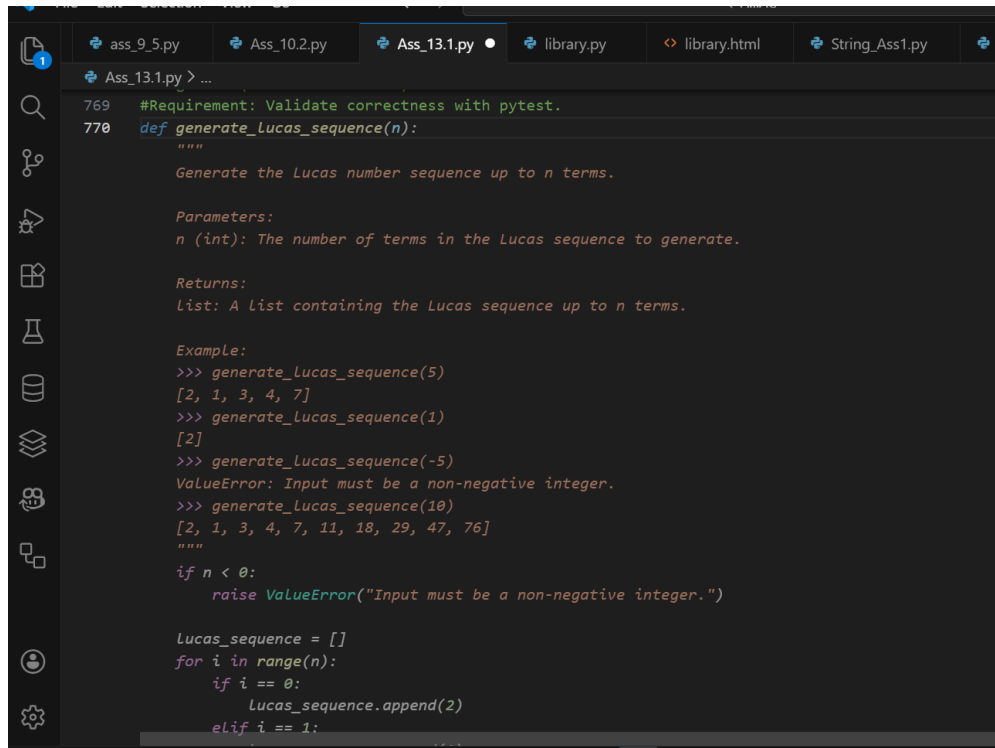
- Function: Generate Lucas sequence up to n terms. (Starts with 2,1, then  $F_n = F_{n-1} + F_{n-2}$ ) Test Cases to Design:
- Normal: 5 → [2, 1, 3, 4, 7]
- Edge: 1 → [2]
- Negative: -5 → Error
- Large: 10 (last element = 76). Requirement: Validate correctness with pytest.

```
#Task 14 (Lucas Number Sequence – Test Case Design)
#Prompt: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.
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
# Example usage
n = int(input("Enter the number of terms: ")) # Get user input for the
```

```

try:
    lucas_sequence = generate_lucas_sequence(n) # Generate the Lucas sequence
    print(lucas_sequence) # Print the generated sequence
except ValueError as e:
    print(e) # Print the error message if input is invalid

```



The screenshot shows a code editor with a dark theme. The top bar displays several open 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 left sidebar contains icons for file explorer, search, and other IDE features. The main editor area shows the following Python code:

```

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:

```

```

File Edit Selection View Go ...
Ass_9.5.py Ass_10.2.py Ass_13.1.py X library.py library.html String_Ass1.py
Ass_13.1.py > generate_lucas_sequence
770 def generate_lucas_sequence(n):
788     [2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
789     """
790     if n < 0:
791         raise ValueError("Input must be a non-negative integer.")
792
793     lucas_sequence = []
794     for i in range(n):
795         if i == 0:
796             lucas_sequence.append(2)
797         elif i == 1:
798             lucas_sequence.append(1)
799         else:
800             next_lucas = lucas_sequence[i-1] + lucas_sequence[i-2]
801             lucas_sequence.append(next_lucas)
802
803     return lucas_sequence
804 # Example usage
805 n = int(input("Enter the number of terms: ")) # Get user input for the
806 try:
807     lucas_sequence = generate_lucas_sequence(n) # Generate the Lucas sequence
808     print(lucas_sequence) # Print the generated sequence
809 except ValueError as e:
810     print(e) # Print the error message if input is invalid
811
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
Enter the number of terms: 5
[2, 1, 3, 4, 7]
PS C:\Users\hbruth\OneDrive\Desktop\A.I.AC>

```



### 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

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

#Prompt: 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.

```
def count_vowels_consonants(s):
```

```
    """
```

Count the number of vowels and consonants in a given string.

Parameters:

s (str): The input string to be analyzed.

Returns:

tuple: A tuple containing the count of vowels and consonants in the format (vowel\_count, consonant\_count).

Example:

```
>>> count_vowels_consonants("hello")
```

```
(2, 3)
```

```
>>> count_vowels_consonants("")
```

```
(0, 0)
```

```
>>> count_vowels_consonants("aeiou")
```

```
(5, 0)
```

```
"""
```

```
vowels = 'aeiouAEIOU'
```

```
vowel_count = sum(1 for char in s if char in vowels)
```

```
consonant_count = sum(1 for char in s if char.isalpha() and char not in vowels)
```

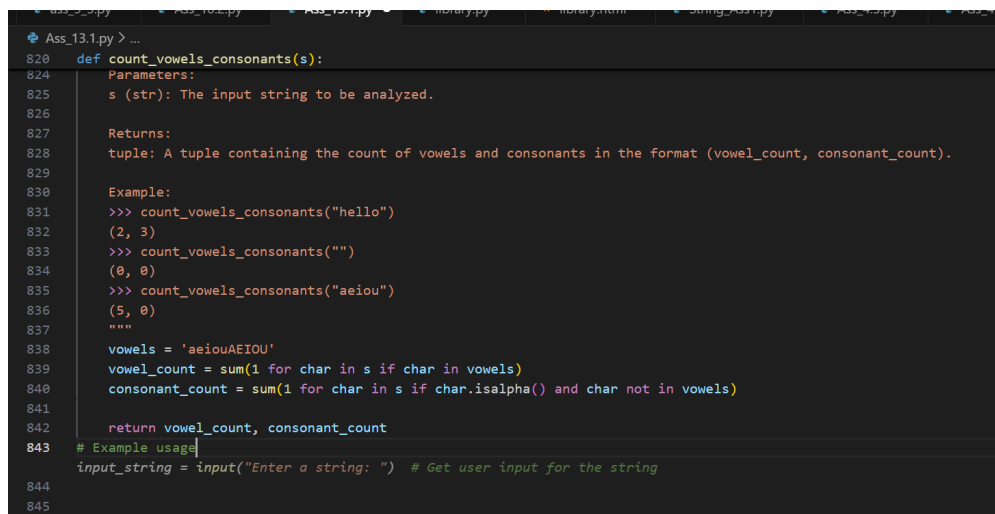
```
return vowel_count, consonant_count
```

# Example usage

```
input_string = input("Enter a string: ") # Get user input for the string
```

```
vowel_count, consonant_count = count_vowels_consonants(input_string) # Count vowels and consonants
```

```
print(f"Vowels: {vowel_count}, Consonants: {consonant_count}") # Print the counts
```



```
Ass_13.1.py > ...
820 def count_vowels_consonants(s):
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     consonant_count = sum(1 for char in s if char.isalpha() and char not in vowels)
841
842     return vowel_count, consonant_count
843 # Example usage
844 input_string = input("Enter a string: ") # Get user input for the string
845
```

```
Ass_13.1.py > ...
817 #Only vowels: "aeiou" → (5,0)
818 #Large: Long text
819 #Requirement: Validate correctness with pytest.
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     consonant_count = len(s) - vowel_count
841     return (vowel_count, consonant_count)
842
843 # Test cases
844 test_cases = ["hello", "world", "python", "programming", "data science", "machine learning", "artificial intelligence", "deep learning", "reinforcement learning", "natural language processing", "computer vision", "robotics", "autonomous systems", "cybersecurity", "cloud computing", "big data", "blockchain", "quantum computing", "space exploration", "biotechnology", "nanotechnology", "nanomedicine", "biomimetics", "synthetic biology", "gene editing", "CRISPR", "mRNA vaccines", "AI ethics", "AI law", "AI policy", "AI governance", "AI accountability", "AI transparency", "AI explainability", "AI interpretability", "AI auditability", "AI reliability", "AI robustness", "AI resilience", "AI adaptability", "AI flexibility", "AI scalability", "AI interoperability", "AI compatibility", "AI portability", "AI reusability", "AI modularity", "AI extensibility", "AI customizability", "AI configurability", "AI adaptability", "AI flexibility", "AI scalability", "AI interoperability", "AI compatibility", "AI portability", "AI reusability", "AI modularity", "AI extensibility", "AI customizability", "AI configurability"]
845
846 # Run tests
847 pytest.main([__file__, "-v"])
848
849 # Main function
850 def main():
851     s = input("Enter a string: ")
852     vowel_count, consonant_count = count_vowels_consonants(s)
853     print(f"Vowels: {vowel_count}, Consonants: {consonant_count}")
854
855 if __name__ == "__main__":
856     main()
```

PROBLEMS OUTPUT **TERMINAL** PORTS DEBUG CONSOLE

```
...
Enter a string: Hruthika
Vowels: 3, Consonants: 5
```