

Lab Assignment-13

Name: D.BHARATH KUMAR

Hallticket:2303A51097

Batch-02

Task Description #1 (Refactoring – Removing Code Duplication)

- Task: Use AI to refactor a given Python script that contains multiple repeated code blocks.
- Instructions:
 - o Prompt AI to identify duplicate logic and replace it with functions or classes.
 - o Ensure the refactored code maintains the same output.
 - o Add docstrings to all functions.
- Sample Legacy Code:

```
# Legacy script with repeated logic  
print("Area of Rectangle:", 5 * 10)  
print("Perimeter of Rectangle:", 2 * (5 + 10))  
print("Area of Rectangle:", 7 * 12)  
print("Perimeter of Rectangle:", 2 * (7 + 12))  
print("Area of Rectangle:", 10 * 15)  
print("Perimeter of Rectangle:", 2 * (10 + 15))
```

- Expected Output:
 - o Refactored code with a reusable function and no duplication.
 - o Well documented code

screenshots:

```
assg_13.py > validate_dimensions
1 def validate_dimensions(length, width):
2     """
3         Validate that dimensions are positive numbers.
4
5     Args:
6         length: The length of the rectangle
7         width: The width of the rectangle
8
9     Returns:
10        bool: True if valid, raises ValueError otherwise
11    """
12    if not isinstance(length, (int, float)) or not isinstance(width, (int, float)):
13        raise ValueError("Dimensions must be numbers")
14    if length <= 0 or width <= 0:
15        raise ValueError("Dimensions must be positive")
16    return True
17
18
19 def calculate_area(length, width):
20     """
21         Calculate the area of a rectangle.
22
23     Args:
24         length: The length of the rectangle
25         width: The width of the rectangle
26
27     Returns:
28         float: The area of the rectangle
29    """
30    validate_dimensions(length, width)
31    return length * width
32
33
34 def calculate_perimeter(length, width):
35     """
36         Calculate the perimeter of a rectangle.
37
38     Args:
39         length: The length of the rectangle
40         width: The width of the rectangle
41
42     Returns:
43         float: The perimeter of the rectangle
44    """
45    validate_dimensions(length, width)
46    return 2 * (length + width)
47
48
49 def print_rectangle_info(length, width):
50     """
51         Print area and perimeter of a rectangle.
52
53     Args:
54         length: The length of the rectangle
55         width: The width of the rectangle
56    """
57    try:
58        area = calculate_area(length, width)
59        perimeter = calculate_perimeter(length, width)
60        print(f"Area of Rectangle: {area}")
61        print(f"Perimeter of Rectangle: {perimeter}")
62    except ValueError as e:
63        print(f"Error: {e}")
64
65
66 # Usage
67 print_rectangle_info(5, 10)
68 print_rectangle_info(7, 12)
69 print_rectangle_info(10, 15)
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_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\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Task Description #2 (Refactoring – Extracting Reusable Functions)

- Task: Use AI to refactor a legacy script where multiple calculations are embedded directly inside the main code block.

- Instructions:

- o Identify repeated or related logic and extract it into reusable functions.
 - o Ensure the refactored code is modular, easy to read, and documented with docstrings.

- Sample Legacy Code:

```
# Legacy script with inline repeated logic

price = 250

tax = price * 0.18

total = price + tax

print("Total Price:", total)

price = 500

tax = price * 0.18

total = price + tax

print("Total Price:", total)
```

- Expected Output:

- o Code with a function calculate_total(price) that can be reused for multiple price inputs.

- o Well documented code

screenshots:

```

39 #refactor the code identify the replicated or related logics from the code extract into the reusable function and use that function in the main code to avoid code repetition and make the code more modular and maintainable
40 def calculate_total_price(price):
41     """
42         Calculate the total price including tax.
43         This function takes a price and calculates the total cost by adding an 18% tax
44         to the original price.
45     Args:
46         price (int or float): The base price before tax. Must be a non-negative number.
47     Returns:
48         float: The total price including 18% tax.
49     Raises:
50         TypeError: If price is not a number (int or float).
51         ValueError: If price is negative.
52     Example:
53         >>> calculate_total_price(100)
54         118.0
55         >>> calculate_total_price(50.5)
56         59.59
57         """
58         if not isinstance(price, (int, float)):
59             raise TypeError("Price must be a number.")
60         if price < 0:
61             raise ValueError("Price must be non-negative.")
62
63         tax = price * 0.18
64         total = price + tax
65         return total
66
67 price = 1000
68 try:
69     total_price = calculate_total_price(price)
70     print("Total Price:", total_price)
71 except (TypeError, ValueError) as e:
72     print(f"Error: {e}")
73 price = 500
74 try:
75     total_price = calculate_total_price(price)
76     print("Total Price:", total_price)
77 except (TypeError, ValueError) as e:
78     print(f"Error: {e}")
79 price = 500
80 try:
81     total_price = calculate_total_price(price)
82     print("Total Price:", total_price)
83 except (TypeError, ValueError) as e:
84     print(f"Error: {e}")

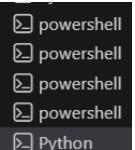
```

Output:

```

Perimeter of Rectangle: 50
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13 (1).py"
Total Price: 1180.0
Total Price: 590.0
Total Price: 590.0
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```



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

Redundant Conditional Logic)

Refactor a Python script that contains repeated if–elif–else grading logic

by implementing a structured, object-oriented solution using a class and a method.

Problem Statement

The given script contains duplicated conditional statements used to assign grades based on student marks. This redundancy violates clean code principles and reduces maintainability.

You are required to refactor the script using a class-based design to improve modularity, reusability, and readability while preserving the original grading logic.

Mandatory Implementation Requirements

1. Class Name: GradeCalculator
2. Method Name: calculate_grade(self, marks)
3. The method must:

- o Accept marks as a parameter.
- o Return the corresponding grade as a string.
- o The grading logic must strictly follow the conditions below:
 - Marks ≥ 90 and $\leq 100 \rightarrow$ "Grade A"
 - Marks $\geq 80 \rightarrow$ "Grade B"
 - Marks $\geq 70 \rightarrow$ "Grade C"
 - Marks $\geq 40 \rightarrow$ "Grade D"
 - Marks $\geq 0 \rightarrow$ "Fail"

Note: Assume marks are within the valid range of 0 to 100.

4. Include proper docstrings for:

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

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

• Given code:

```

marks = 85

if marks >= 90:
    print("Grade A")

elif marks >= 75:
    print("Grade B")

else:
    print("Grade C")

marks = 72

if marks >= 90:
    print("Grade A")

elif marks >= 75:
    print("Grade B")

else:
    print("Grade C")

```

Expected Output:

- Define a class named GradeCalculator.

- Implement a method `calculate_grade(self, marks)` inside the class.
- Create an object of the class.
- Call the method for different student marks.
- Print the returned grade values.

Screenshots:

```

68  #refactor the given code that contains if elif else grading statements by implementing a structured object orientated solution using class and a method
69  # Marks ≥ 80 → "Grade B"
70  # Marks ≥ 70 → "Grade C"
71  # Marks ≥ 40 → "Grade D"
72  # Marks ≥ 0 → "Fail"
73  #include proper doctring and error handling for invalid input and class and method (with parameters) to calculate the grade based on the marks provided
74  class GradeCalculator:
75      def __init__(self, marks):
76          self.marks = marks
77
78      def calculate_grade(self):
79          if not isinstance(self.marks, (int, float)):
80              raise TypeError("Marks must be a number.")
81          if self.marks < 0 or self.marks > 100:
82              raise ValueError("Marks must be between 0 and 100.")
83
84          if self.marks >= 90:
85              return "Grade A"
86          elif self.marks >= 80:
87              return "Grade B"
88          elif self.marks >= 70:
89              return "Grade C"
90          elif self.marks >= 40:
91              return "Grade D"
92          else:
93              return "Fail"
94
95      marks = 85
96      if marks >= 90:
97          print("Grade A")
98      elif marks >= 75:
99          print("Grade B")
100     else:
101         print("Grade C")

```

```

71 class GradeCalculator:
72     """GradeCalculator Module
73     A utility module for calculating student grades based on numerical marks.
74     Classes:
75         GradeCalculator: Evaluates and assigns letter grades to student marks.
76     Example:
77         >>> calculator = GradeCalculator(85)
78         >>> calculator.calculate_grade()
79         'Grade B'
80     Note:
81         This module assumes marks are on a 0-100 scale. Grades are assigned
82         according to the following scale:
83         - 90-100: Grade A
84         - 80-89: Grade B
85         - 70-79: Grade C
86         - 40-69: Grade D
87         - 0-39: Fail
88     Class to calculate student grades based on marks."""
89
90     def __init__(self, marks):
91         """Initialize with marks value."""
92         self.marks = marks
93
94     def calculate_grade(self):
95         """Calculate and return grade based on marks with proper validation.
96         Raises:
97             TypeError: If marks is not a number.
98             ValueError: If marks is out of valid range (0-100).
99
100        """
101        if not isinstance(self.marks, (int, float)):
102            raise TypeError("Marks must be a number.")
103        if self.marks < 0 or self.marks > 100:
104            raise ValueError("Marks must be between 0 and 100.")
105
106        if self.marks >= 90:
107            return "Grade A"
108        elif self.marks >= 80:
109            return "Grade B"
110        elif self.marks >= 70:
111            return "Grade C"
112        elif self.marks >= 40:
113            return "Grade D"
114        else:
115            return "Fail"
116
117
118 # Usage with error handling
119 test_marks = [85, 72, 35, 95]
120
121 for marks in test_marks:
122     try:
123         grade_calculator = GradeCalculator(marks)
124         grade = grade_calculator.calculate_grade()
125         print(f"Marks: {marks} → {grade}")
126     except (TypeError, ValueError) as e:
127         print(f"Error: {e}")

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13 (1).py"
Marks: 85 → Grade B
Marks: 72 → Grade C
Marks: 35 → Fail
Marks: 95 → Grade A
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

Task Description #4 (Refactoring – Converting Procedural Code to

Functions)

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

Instructions:

- o Identify input, processing, and output sections.

- o Convert each into a separate function.
- o Improve code readability without changing behavior.

- Sample Legacy Code:

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

square = num * num

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

- Expected Output:

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

Screenshots:

```
108 #identify input ,processing and output sections and convert them into a separate function for each section like and add docstrings to each function
109 def get_input():
110     """Function to get input from the user."""
111     try:
112         num = int(input("Enter number: "))
113         return num
114     except ValueError:
115         print("Invalid input. Please enter a valid integer.")
116         return None
117 def process_input(num):
118     """Function to process the input and calculate the square."""
119     if num is not None:
120         return num ** 2
121     return None
122 def display_output(result):
123     """Function to display the output."""
124     if result is not None:
125         print(f"The square of the number is: {result}")
126     else:
127         print("No result to display.")
128
129
130
131
132
133
134
```

```
107 #identify input ,processing and output sections and convert them into a separate function for each section like and add docstrings to each function
108 def get_input():
109     """Function to get input from the user."""
110     try:
111         num = int(input("Enter number: "))
112         return num
113     except ValueError:
114         print("Invalid input. Please enter a valid integer.")
115         return None
116 def process_input(num):
117     """Function to process the input and calculate the square."""
118     if num is not None:
119         return num ** 2
120     return None
121 def display_output(result):
122     """Function to display the output."""
123     if result is not None:
124         print(f"The square of the number is: {result}")
125     else:
126         print("No valid result to display.")
127 def main():
128     """Main function to orchestrate the input, processing, and output."""
129     num = get_input()
130     result = process_input(num)
131     display_output(result)
132
133 if __name__ == "__main__":
134     main()
```

Output:

```
HRS: 55 % grade: A
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13 (1).py"
Enter number: 100
The square of the number is: 10000
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

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:

A class like EmployeeSalaryCalculator with methods and attributes.

Screenshots:

```
132 #refactor the code into the procedural code into a class -based design main focus on the object ooriented line encapsulation  
133  
134 class SalaryCalculator:  
135     """Class to calculate salary and tax with encapsulation."""  
136  
137     def __init__(self, salary):  
138         """Initialize SalaryCalculator with salary amount.  
139  
140         Args:  
141             | salary: The base salary amount  
142         """  
143         self._salary = salary  
144  
145     def _validate_salary(self):  
146         """Validate that salary is a positive number.  
147  
148         Raises:  
149             | ValueError: If salary is not positive or not a number  
150         """  
151         if not isinstance(self._salary, (int, float)):  
152             raise ValueError("Salary must be a number.")  
153         if self._salary < 0:  
154             raise ValueError("Salary must be non-negative.")  
155  
156     def calculate_tax(self):  
157         """Calculate tax on the salary.  
158  
159         Returns:  
160             | float: Tax amount (20% of salary)  
161         """  
162         self._validate_salary()  
163         return self._salary * 0.2  
164  
165     def calculate_net_salary(self):  
166         """Calculate net salary after tax deduction.  
167  
168         Returns:  
169             | float: Net salary (salary - tax)  
170         """  
171         tax = self.calculate_tax()  
172         return self._salary - tax  
173  
174     def display_salary_info(self):  
175         """Display salary breakdown."""  
176         try:  
177             tax = self.calculate_tax()  
178             net = self.calculate_net_salary()  
179             print(f"Gross Salary: {self._salary}")  
180             print(f"Tax (20%): {tax}")  
181             print(f"Net Salary: {net}")  
182         except ValueError as e:  
183             print(f"Error: {e}")  
184  
185 # Usage  
186 salary_calculator = SalaryCalculator(50000)  
187 salary_calculator.display_salary_info()
```

Output:

```
THE SQUARE OF THE NUMBER IS: 10000
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13 (1).py"
Gross Salary: 50000
Tax (20%): 10000.0
Net Salary: 40000.0
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> 
```

Ln 168, Col 40 Spaces: 4 UTF-8 CR

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

screenshots:

```
172 # refactor the code to inefficient liner searches using appriate data and focus on the time complexity and data structures edit tge code to the b
173
174 # Using a set for O(1) average-case lookup instead of O(n) list search
175 users = {"admin", "guest", "editor", "viewer"}
176 name = input("Enter username: ")
177 found = name in users
178 print("Access Granted" if found else "Access Denied")
179 
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe "c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13 (1).py"
Enter username: admin
Access Granted
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> 
```

Task 7 – Refactoring the Library Management System

Problem Statement

You are provided with a poorly structured Library Management script that:

- Contains repeated conditional logic
- Does not use reusable functions
- Lacks documentation
- Uses print-based procedural execution
- Does not follow modular programming principles

Your task is to refactor the code into a proper format

1. Create a module library.py with functions:

- o add_book(title, author, isbn)
- o remove_book(isbn)
- o search_book(isbn)

2. Insert triple quotes under each function and let Copilot complete the docstrings.

3. Generate documentation in the terminal.

4. Export the documentation in HTML format.

5. Open the file in a browser.

Given Code

```
# Library Management System (Unstructured Version)

# This code needs refactoring into a proper module with documentation.

library_db = {}

# Adding first book

title = "Python Basics"

author = "John Doe"

isbn = "101"

if isbn not in library_db:

    library_db[isbn] = {"title": title, "author": author}

    print("Book added successfully.")

else:

    print("Book already exists.")

# Adding second book (duplicate logic)

title = "AI Fundamentals"

author = "Jane Smith"
```

```
isbn = "102"

if isbn not in library_db:
    library_db[isbn] = {"title": title, "author": author}
    print("Book added successfully.")

else:
    print("Book already exists.")

# Searching book (repeated logic structure)

isbn = "101"

if isbn in library_db:
    print("Book Found:", library_db[isbn])

else:
    print("Book not found.")

# Removing book (again repeated pattern)

isbn = "101"

if isbn in library_db:
    del library_db[isbn]
    print("Book removed successfully.")

else:
    print("Book not found.")

# Searching again

isbn = "101"

if isbn in library_db:
    print("Book Found:", library_db[isbn])

else:
    print("Book not found.")
```

screenshots:

```

180 """
181 Library Management System - Refactored Module
182 Provides functions to manage a library database with proper documentation.
183 """
184
185 library_db = {}
186
187 def add_book(title, author, isbn):
188     """
189     Add a new book to the library database.
190
191     Args:
192         title (str): The title of the book.
193         author (str): The author of the book.
194         isbn (str): The ISBN of the book (unique identifier).
195
196     Returns:
197         bool: True if book added successfully, False if book already exists.
198
199     Raises:
200         ValueError: If any parameter is empty or invalid.
201     """
202     if not title or not author or not isbn:
203         raise ValueError("Title, author, and ISBN cannot be empty.")
204
205     if isbn not in library_db:
206         library_db[isbn] = {"title": title, "author": author}
207         print("Book added successfully.")
208         return True
209     else:
210         print("Book already exists.")
211         return False
212
213 def remove_book(isbn):
214     """
215     Remove a book from the library database by ISBN.
216
217     Args:
218         isbn (str): The ISBN of the book to remove.
219
220     Returns:
221         bool: True if book removed successfully, False if book not found.
222     """
223     if isbn in library_db:
224         del library_db[isbn]
225         print("Book removed successfully.")
226         return True
227     else:
228         print("Book not found.")
229         return False
230
231 def search_book(isbn):
232     """
233     Search for a book in the library database by ISBN.
234
235     Args:
236         isbn (str): The ISBN of the book to search.
237
238     Returns:
239         dict: Book details (title, author) if found, None otherwise.
240     """
241     if isbn in library_db:
242         print("Book Found:", library_db[isbn])
243         return library_db[isbn]
244     else:
245         print("Book not found.")
246         return None
247
248 # Usage
249 if __name__ == "__main__":
250     add_book("Python Basics", "John Doe", "101")
251     add_book("AI Fundamentals", "Jane Smith", "102")
252     search_book("101")
253     remove_book("101")
254     search_book("101")
255

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/users/bharath/Desktop/AIAC/library.py
Book added successfully.
Book added successfully.
Book Found: {'title': 'Python Basics', 'author': 'John Doe'}
Book removed successfully.
Book not found.
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

Functions

`add_book(title, author, isbn)`

This function creates a new book entry in the library database using the provided bibliographic information. It validates that all required parameters are supplied and checks for duplicate entries to maintain data integrity.

`title (str):` The title of the book. Must be a non-empty string representing the full name of the publication.

`author (str):` The author of the book. Must be a non-empty string containing the name(s) of the book's author(s).

`isbn (str):` The International Standard Book Number of the book. Must be a non-empty string that serves as a unique identifier for the book in the database. This acts as the primary key to prevent duplicate entries.

`bool:` True if the book was successfully added to the database, indicating a new entry was created. False if a book with the same ISBN already exists in the database, indicating the operation was rejected to prevent duplicate entries.

`ValueError:` If any of the required parameters (title, author, or isbn) is empty, None, or evaluates to a falsy value. This ensures data quality by preventing incomplete book records from being added to the database.

Examples:

```
>>> add_book("The Great Gatsby", "F. Scott Fitzgerald", "978-0-7432-7356-5")
```

Book added successfully.

True

```
>>> add_book("The Great Gatsby", "F. Scott Fitzgerald", "978-0-7432-7356-5")
```

Book already exists.

False

```
>>> add_book("", "F. Scott Fitzgerald", "978-0-7432-7356-5")
```

`ValueError:` Title, author, and ISBN cannot be empty.

`remove_book(isbn)`

Remove a book from the library database by ISBN.

Args:

`isbn (str):` The ISBN of the book to remove.

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

```

screenshots:

```

226 # refactor the code as per the • Refactor into a clean reusable function (generate_fibonacci)., Add docstrings and test cases., Compare AI-refactored
227
228 def generate_fibonacci(n):
229     """
230         Generate a Fibonacci sequence up to n terms.
231
232     Args:
233         n (int): The number of terms in the Fibonacci sequence.
234
235     Returns:
236         list: A list containing the first n Fibonacci numbers.
237     """
238     if n <= 0:
239         return []
240     elif n == 1:
241         return [0]
242     elif n == 2:
243         return [0, 1]
244
245     fib_sequence = [0, 1]
246     for i in range(2, n):
247         fib_sequence.append(fib_sequence[i-1] + fib_sequence[i-2])
248
249     return fib_sequence
250
251 # Test the function
252 n = int(input("Enter limit: "))
253 fib_numbers = generate_fibonacci(n)
254 for num in fib_numbers:
255     print(num)

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter limit: 5
0
1
1
2
3
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

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")
```

screenshots:

```

250 # refactor the code as per the instructions Refactor into is_prime(n) and is_twin_prime(p1, p2). Add docstrings and optimize.
251 def is_prime(n):
252     """
253     Check if a number is prime.
254
255     Args:
256         n (int): The number to check.
257
258     Returns:
259         bool: True if the number is prime, False otherwise.
260     """
261     if n < 2:
262         return False
263     for i in range(2, int(n**0.5) + 1):
264         if n % i == 0:
265             return False
266     return True
267
268 def is_twin_prime(p1, p2):
269     """
270     Check if two numbers are twin primes.
271
272     Args:
273         p1 (int): First number.
274         p2 (int): Second number.
275
276     Returns:
277         bool: True if both numbers are prime and their difference is 2, False otherwise.
278     """
279     return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
280
281 # Generate a list of twin primes in a given range
282 def generate_twin_primes(start, end):
283     """
284     Generate a list of twin primes in a given range.
285
286     Args:
287         start (int): Start of the range (inclusive).
288         end (int): End of the range (inclusive).
289
290     Returns:
291         list: A list of tuples representing twin primes in the given range.
292     """
293     twin_primes = []
294     for i in range(start, end - 1):
295         if is_twin_prime(i, i + 2):
296             twin_primes.append((i, i + 2))
297     return twin_primes
298
299 # Example usage
300 a = 11
301 b = 13
302 f1 = 0
303 if is_twin_prime(a, b):
304     print(f"{a} and {b} are twin primes.")
305 else:
306     print(f"{a} and {b} are not twin primes.")
307 start_range = 1
308 end_range = 100
309 twin_prime_list = generate_twin_primes(start_range, end_range)
310 print(f"Twin primes between {start_range} and {end_range}: {twin_prime_list}")
311
312

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
11 and 13 are twin primes.
Twin primes between 1 and 100: [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

Task 10 – Refactoring the Chinese Zodiac Program

Objective

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

The current program reads a year from the user and prints the corresponding Chinese Zodiac sign. However, the implementation contains repetitive conditional logic, lacks modular design, and does not follow clean coding principles.

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

Chinese Zodiac Cycle (Repeats Every 12 Years)

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

```
# Chinese Zodiac Program (Unstructured Version)
```

```
# This code needs refactoring.
```

```
year = int(input("Enter a year: "))

if year % 12 == 0:
    print("Monkey")
elif year % 12 == 1:
    print("Rooster")
elif year % 12 == 2:
    print("Dog")
elif year % 12 == 3:
    print("Pig")
elif year % 12 == 4:
    print("Rat")
elif year % 12 == 5:
    print("Ox")
elif year % 12 == 6:
    print("Tiger")
```

```

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.

Screenshots:

```

285
286 #refactor the code as per the instructions 1. Create a reusable function: get_zodiac(year)2. Replace the if-elif chain with a cleaner struc
287
288 def get_zodiac(year):
289     """Return the zodiac sign for a given year."""
290     zodiacs = [
291         "Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox",
292         "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Goat"
293     ]
294     return zodiacs[year % 12]
295
296 def main():
297     try:
298         year = int(input("Enter a year: "))
299         zodiac_sign = get_zodiac(year)
300         print(f"The zodiac sign for the year {year} is: {zodiac_sign}")
301     except ValueError:
302         print("Invalid input. Please enter a valid integer for the year.")
303
304 if __name__ == "__main__":
305     main()

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/u
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter a year: 2020
The zodiac sign for the year 2020 is: Rat
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/u
sers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter a year: 2024
The zodiac sign for the year 2024 is: Dragon
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

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:
 - o Accept an integer parameter.
 - o Return True if the number is divisible by the sum of its digits.
 - o Return False otherwise.
3. Separate user input from core logic.
4. Add proper docstrings.
5. Improve readability and maintainability.

Screenshots:

```
303 # refactor the code as per the instructions 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.
304 def is_harshad(number):
305     """
306         Check if a number is a Harshad number.
307
308     A Harshad number is an integer that is divisible by the sum of its digits.
309
310     Args:
311         number (int): The number to check.
312     Returns:
313         bool: True if the number is a Harshad number, False otherwise.
314     """
315     if number <= 0:
316         return False
317
318     temp = number
319     sum_digits = 0
320
321     while temp > 0:
322         digit = temp % 10
323         sum_digits += digit
324         temp //= 10
325
326     return sum_digits != 0 and number % sum_digits == 0
327 def main():
328     try:
329         num = int(input("Enter a number: "))
330         if is_harshad(num):
331             print("True")
332         else:
333             print("False")
334     except ValueError:
335         print("Invalid input. Please enter a valid integer.")
336     if __name__ == "__main__":
337         main()
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python
ers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter a number: 15
False
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\pytho
ers/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter a number: 20
True
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Task 12 – Refactoring the Factorial Trailing Zeros Program

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

The program calculates the number of trailing zeros in $n!$ (factorial of n).

Problem Statement

The current implementation:

- Calculates the full factorial (inefficient for large n)
- Mixes input handling with business logic
- Uses print statements instead of return values
- Lacks modular structure and documentation

You must refactor the code to improve efficiency, readability, and maintainability.

```
# Factorial Trailing Zeros (Unstructured Version)

n = int(input("Enter a number: "))

fact = 1

i = 1

while i <= n:

    fact = fact * i

    i = i + 1

count = 0

while fact % 10 == 0:

    count = count + 1

    fact = fact // 10

print("Trailing zeros:", count)
```

You must:

1. Create a reusable function: `count_trailing_zeros(n)`
2. The function must:
 - o Accept a non-negative integer n .
 - o Return the number of trailing zeros in $n!$.
3. Do NOT compute the full factorial.
4. Use an optimized mathematical approach (count multiples of 5).
5. Add proper docstrings.
6. Separate user interaction from core logic.
7. Handle edge cases (e.g., negative numbers, zero).

Screenshots:

```
329 #REFACTOR THE CODE AS PER THE INSTRUCTIONS 1. Create a reusable function: count_trailing_zeros(n). The function must: Accept a non-negative integer n. Return the number of trailing zeros in n.
330 def count_trailing_zeros(n):
331     if n < 0:
332         raise ValueError("Input must be a non-negative integer.")
333
334     count = 0
335     power_of_5 = 5
336
337     while n >= power_of_5:
338         count += n // power_of_5
339         power_of_5 *= 5
340
341     return count
342
343 def main():
344     try:
345         n = int(input("Enter a non-negative integer: "))
346         zeros = count_trailing_zeros(n)
347         print(f"The number of trailing zeros in {n}! is: {zeros}")
348     except ValueError as e:
349         print(f"Error: {e}")
350
351 if __name__ == "__main__":
352     main()
353
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/Users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
Enter a non-negative integer: 15
The number of trailing zeros in 15! is: 3
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```

Ln 101, Col 2 Spaces: 4 UTF-8 CRLF [] Python 

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 $\rightarrow 6/2 = 3$)
- 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]

Screenshots:

```

354 """
355 """
356 We need to write a function that:
357 • Takes an integer n as input.
358 • Generates the Collatz sequence (also called the 3n+1 sequence).
359 • The rules are:
360 o If n is even → next = n / 2.
361 o If n is odd → next = 3n + 1.
362 • Repeat until we reach 1.
363 • Return the full sequence as a list.
364
365 """
366 def generate_collatz_sequence(n):
367     """
368         Generate the Collatz sequence for a given positive integer.
369
370     Args:
371         n: A positive integer
372
373     Returns:
374         A list representing the Collatz sequence
375
376     Raises:
377         ValueError: If n is not a positive integer
378     """
379     if not isinstance(n, int) or n <= 0:
380         raise ValueError("Input must be a positive integer.")
381
382     sequence = []
383
384     while n != 1:
385         sequence.append(n)
386         if n % 2 == 0:
387             n //= 2
388         else:
389             n = 3 * n + 1
390
391     sequence.append(1)
392     return sequence
393
394 def main():
395     try:
396         n = int(input("Enter a positive integer: "))
397         collatz_sequence = generate_collatz_sequence(n)
398         print(f"Collatz sequence for {n}: {collatz_sequence}")
399     except ValueError as e:
400         print(f"Error: {e}")
401
402 if __name__ == "__main__":
403     main()
404
405 # Test cases
406 assert generate_collatz_sequence(6) == [6, 3, 10, 5, 16, 8, 4, 2, 1]
407 assert generate_collatz_sequence(19) == [19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
408 assert generate_collatz_sequence(1) == [1]
409
410 try:
411     generate_collatz_sequence(0)
412     assert False, "Should raise ValueError"
413 except ValueError:
414     pass
415
416 try:
417     generate_collatz_sequence(-5)
418     assert False, "Should raise ValueError"
419 except ValueError:
420     pass
421
422 print("All tests passed!")

```

Output:

```

PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/users/bharath/oneDrive/pictures/Desktop/AIAC/assg_13.py
Enter a positive integer: 200
Collatz sequence for 200: [200, 100, 50, 25, 76, 38, 19, 58, 29, 88, 44, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1]
All tests passed!
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

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.

Screenshots:

```

412
413     """
414     • Function: Generate Lucas sequence up to n terms.
415     (Starts with 2,1, then  $F_n = F_{n-1} + F_{n-2}$ )
416     • Test Cases to Design:
417     • Normal: 5 → [2, 1, 3, 4, 7]
418     • Edge: 1 → [2]
419     • Negative: -5 → Error
420     • Large: 10 (last element = 76).
421     • Requirement: Validate correctness with pytest.
422
423     """
424
425     def generate_lucas_sequence(n):
426         if n <= 0:
427             raise ValueError("Input must be a positive integer.")
428
429         sequence = []
430         a, b = 2, 1
431
432         for _ in range(n):
433             sequence.append(a)
434             a, b = b, a + b
435
436         return sequence
437
438     # Test cases
439     assert generate_lucas_sequence(5) == [2, 1, 3, 4, 7]
440     assert generate_lucas_sequence(1) == [2]
441     try:
442         generate_lucas_sequence(-5)
443         assert False, "Should raise ValueError"
444     except ValueError:
445         pass
446     assert generate_lucas_sequence(10) == [2, 1, 3, 4, 7, 11, 18, 29, 47, 76]
447     print("All tests passed!")

```

Output:

```

All tests passed!
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
All tests passed!
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>

```

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

- Function: Count vowels and consonants in string.
- Test Cases to Design:
- Normal: "hello" → (2,3)
- Edge: "" → (0,0)
- Only vowels: "aeiou" → (5,0)

Large: Long text

- Requirement: Validate correctness with pytest.

Screenshots:

```
447 """
448     • Function: Count vowels and consonants in string.
449     • Test Cases to Design:
450     • Normal: "hello" → (2,3)
451     • Edge: "" → (0,0)
452     • Only vowels: "aeiou" → (5,0)
453 Large: Long text
454     • Requirement: Validate correctness with pytest.
455
456 """
457 def count_vowels_consonants(s):
458     vowels = set("aeiouAEIOU")
459     vowel_count = 0
460     consonant_count = 0
461
462     for char in s:
463         if char.isalpha():
464             if char in vowels:
465                 vowel_count += 1
466             else:
467                 consonant_count += 1
468
469     return vowel_count, consonant_count
470 # Test cases
471 assert count_vowels_consonants("hello") == (2, 3)
472 assert count_vowels_consonants("") == (0, 0)
473 assert count_vowels_consonants("aeiou") == (5, 0)
474 assert count_vowels_consonants("The quick brown fox jumps over the lazy dog") == (11, 24)
475 print("All tests passed!")
```

Output:

```
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC> & C:\Users\BHARATH\AppData\Local\Programs\Python\Python313\python.exe c:/users/BHARATH/OneDrive/Pictures/Desktop/AIAC/assg_13.py
All tests passed!
PS C:\Users\BHARATH\OneDrive\Pictures\Desktop\AIAC>
```