

Aissignment-13.1

2303A51663

Batch-23

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

```
# 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))
# Refactored the above code with a reusable function and no duplication and well documented code

def calculate_rectangle_properties(length, width):
    """
    Calculate the area and perimeter of a rectangle.

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

    Returns:
    tuple: A tuple containing the area and perimeter of the rectangle.
    """
    area = length * width
    perimeter = 2 * (length + width)
    return area, perimeter

# List of rectangles with their lengths and widths
rectangles = [(5, 10), (7, 12), (10, 15)]
# Calculate and print properties for each rectangle
for length, width in rectangles:
    area, perimeter = calculate_rectangle_properties(length, width)
    print(f"Area of Rectangle (length={length}, width={width}): {area}")
    print(f"Perimeter of Rectangle (length={length}, width={width}): {perimeter}")
    print() # Add a newline for better readability between rectangles
```

```
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> & C:\Use
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> & C:\Use
Area of Rectangle: 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
Area of Rectangle (length=5, width=10): 50
Perimeter of Rectangle: 30
Area of Rectangle: 84
Perimeter of Rectangle: 38
Area of Rectangle: 150
Perimeter of Rectangle: 50
Area of Rectangle (length=5, width=10): 50
Perimeter of Rectangle (length=5, width=10): 30

Area of Rectangle (length=7, width=12): 84
Perimeter of Rectangle (length=7, width=12): 38

Area of Rectangle (length=10, width=15): 150
Perimeter of Rectangle (length=10, width=15): 50

PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> 
```

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:

Week7

-

Monda

y

```
# Legacy script with inline repeated logic
```

```
price = 250
```

```
tax = price * 0.18
```

```
total = price + tax
```

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

```
price = 500
```

```
tax = price * 0.18
```

```
total = price + tax
```

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

- Expected Output:

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

- o Well documented code

```
# 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)

#refactored the above code with a function calculate_total_price that can be reused for multiple prices and well documented code
def calculate_total_price(price):
    """
    Calculate the total price of an item, including tax.
    Parameters:
    price (float): The price of the item.
    Returns:
    float: The total price, including tax.
    """
    tax = price * 0.18
    total = price + tax
    return total

# List of prices to calculate total price for
prices = [250, 500]
# Calculate and print total price for each price
for price in prices:
    total_price = calculate_total_price(price)
    print(f"Total Price (price={price}): {total_price}")
    print() # Add a newline for better readability between prices
```

```
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> ^C
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> & C:\Users\Aishwarya\OneDrive\Desktop\AI LAB>
Total Price: 295.0
Total Price: 590.0
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> & C:\Users\Aishwarya\OneDrive\Desktop\AI LAB>
Total Price: 295.0
Total Price: 590.0
Total Price: 295.0
Total Price: 590.0
Total Price (price=250): 295.0

Total Price (price=500): 590.0

PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB>
```

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.

```

#Refactoring Using Classes and Methods (Eliminating Redundant Conditional Logic)
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.

class GradeCalculator:
    def calculate_grade(self, marks):
        """
        Calculate the grade based on the marks obtained.
        Parameters:
        marks (float): The marks obtained by the student.
        Returns:
        str: The grade corresponding to the marks.
        """
        if not isinstance(marks, (int, float)):
            raise ValueError("Marks must be a number.")
        if marks < 0 or marks > 100:
            raise ValueError("Marks must be between 0 and 100.")

        if marks >= 90:
            return 'A'
        elif marks >= 80:
            return 'B'
        elif marks >= 70:
            return 'C'
        elif marks >= 60:
            return 'D'
        else:
            return 'F'

# Create an object of the GradeCalculator class
grade_calculator = GradeCalculator()
# Call the method for different student marks and print the returned grade values
student_marks = [95, 85, 75, 65, 55]
for marks in student_marks:
    grade = grade_calculator.calculate_grade(marks)
    print(f"Marks: {marks}, Grade: {grade}")
    # Add a new line for better readability between students
# display the output in a class name grade calculator of student marks and grade values
print("Class Name: GradeCalculator")
print("Student Marks and Grade Values:")
for marks in student_marks:
    grade = grade_calculator.calculate_grade(marks)
    print(f"Marks: {marks}, Grade: {grade}")

```

```

Class Name: GradeCalculator
Student Marks and Grade Values:
Marks: 95, Grade: A
Marks: 85, Grade: B
Marks: 75, Grade: C
Marks: 65, Grade: D
Marks: 55, Grade: F

```

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().

```

num = int(input("Enter number: "))
square = num * num
print("Square:", square)
#refactor the above code Use AI to refactor procedural input-processing logic into function
def calculate_square(num):
    """
    Calculate the square of a number.

    Parameters:
    num (int): The number to be squared.

    Returns:
    int: The square of the input number.
    """

    return num * num
# Get user input and calculate the square
try:
    num = int(input("Enter number: "))
    square = calculate_square(num)
    print("Square:", square)
except ValueError:
    print("Invalid input. Please enter a valid integer.")
#display result
print(f"Input Number: {num}, Square: {square}")

```

```

C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> python ex1.py
Enter number: 5
Square: 25
Enter number: 10
Square: 100
Input Number: 10, Square: 100
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB>

```

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.

```
salary = 50000  
tax = salary * 0.2  
net = salary - tax  
print(net)  
#refactor procedural code into a class-based design. with attributes and methods to calculate  
class SalaryCalculator:  
    def __init__(self, salary):  
        self.salary = salary  
  
    def calculate_tax(self):  
        """  
        Calculate the tax based on the salary.  
        Returns:  
        float: The calculated tax amount.  
        """  
        return self.salary * 0.2  
  
    def calculate_net_salary(self):  
        """  
        Calculate the net salary after deducting tax.  
        Returns:  
        float: The calculated net salary.  
        """  
        tax = self.calculate_tax()  
        net_salary = self.salary - tax  
        return net_salary  
  
# Create an object of the SalaryCalculator class  
salary_calculator = SalaryCalculator(50000)  
# Calculate and print tax and net salary  
tax = salary_calculator.calculate_tax()  
net_salary = salary_calculator.calculate_net_salary()  
print("Tax:", tax)  
print("Net Salary:", net_salary)
```

```
40000.0  
Tax: 10000.0  
Net Salary: 40000.0  
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB>
```

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

```
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")
#Refactor inefficient linear searches using appropriate data Use of sets or dictionaries w
users = {"admin", "guest", "editor", "viewer"} # Using a set for O(1) average time complex
name = input("Enter username: ")
if name in users:
    print("Access Granted")
else:
    print("Access Denied")
print("Access Granted" if name in users else "Access Denied")
```

```
Enter username: admin guest editor viewer
Access Denied
Enter username: admin
Access Granted
Access Granted
PS C:\Users\Aishwarya\OneDrive\Desktop\AI LAB> █
```

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.")

```

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.")
```

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 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")
```

The screenshot shows a code editor interface with a dark theme. The top navigation bar includes tabs for "library.py X", "task 1.py", "task2.py", "task3.py", "unit test.py 2", "documentationexample.py", and "math_". The main code editor area contains the following Python script:

```
❶ #library mangement system
❷ def add_book(title,author,isbn):
❸     book = {
❹         'title': title,
❺         'author': author,
❻         'isbn': isbn
❼     }
➋     return book
⌽ def remove_book(book_list, isbn):
⌾     for book in book_list:
⌿         if book['isbn'] == isbn:
⌿             book_list.remove(book)
⌿             return True
⌽     return False
⌽ def search_book(book_list, title):
⌾     for book in book_list:
⌿         if book['title'].lower() == title.lower():
⌿             return book
⌽     return None
⌽ print(add_book("The Great Gatsby", "F. Scott Fitzgerald", "978-0743273565"))
⌽ books = []
⌽ books.append(add_book("To Kill a Mockingbird", "Harper Lee", "978-0061120084"))
⌽ books.append(add_book("1984", "George Orwell", "978-0451524935"))
⌽ print(search_book(books, "1984"))
⌽ print(remove_book(books, "978-0061120084"))
⌽ print(books)
```

Below the code editor is a terminal window showing the execution of the script and its output:

```
PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
True
[{'title': '1984', 'author': 'George Orwell', 'isbn': '978-0451524935'}]
❸ PS C:\Users\HP\OneDrive\Desktop\AI 2026> python -m pydoc -w library
{'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald', 'isbn': '978-0743273565'}
{'title': '1984', 'author': 'George Orwell', 'isbn': '978-0451524935'}
True
[{'title': '1984', 'author': 'George Orwell', 'isbn': '978-0451524935'}]
wrote library.html
❸ PS C:\Users\HP\OneDrive\Desktop\AI 2026> python -m pydoc -p 8080
[WinError 10013] An attempt was made to access a socket in a way forbidden by its access permissions
❸ PS C:\Users\HP\OneDrive\Desktop\AI 2026> python -m pydoc -p 1234
Server ready at http://localhost:1234/
Server commands: [b]rowser, [q]uit
server> b
server> {'title': 'The Great Gatsby', 'author': 'F. Scott Fitzgerald', 'isbn': '978-0743273565'}
{'title': '1984', 'author': 'George Orwell', 'isbn': '978-0451524935'}
True
[{'title': '1984', 'author': 'George Orwell', 'isbn': '978-0451524935'}]
```

[index](#)
library <c:\users\hp\onedrive\desktop\ai 2026\library.py>

#library mangement system

Functions

```
add_book(title, author, isbn)
#library mangement system

remove_book(book_list, isbn)

search_book(book_list, title)
```

Data

```
books = [ {'author': 'George Orwell', 'isbn': '978-0451524935', 'title': '1984'}]
```

```
Python 3.13.12 [tags/v3.13.12:1cbe481, MSC v.1944 64 bit (AMD64)]  
Windows-11
```

library

```
#library mangement system
```

Functions

```
add_book(title, author, isbn)  
    #library mangement system  
  
remove_book(book_list, isbn)  
  
search_book(book_list, title)
```

Data

```
books = [ {'author': 'George Orwell', 'isbn': '978-0451524935', 'title': '1984'} ]
```

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
```

The screenshot shows a Python code editor with a file named `lab 13 .6.py`. The code defines a generator function `fibonacci_generator` that yields the first `n` numbers in the Fibonacci sequence. A user enters a limit of 12, and the generator is used to print the sequence.

```
1 Welcome
2 Untitled-3.py
3 lab test 12.py
4 12.b.py
5 examples of test cases.py
6 demo1.py
7 testcasedemo.py
8 ass 5.1 and 6.py
9 ass 7.4.py
10 demo 9.py
11 demo2.py
12
13 # lab 13 .6.py > _
14 def fibonacci_generator(n):
15     """
16         This function generates the Fibonacci sequence up to the nth number.
17         Parameters:
18             n (int): The number of Fibonacci numbers to generate.
19         Returns:
20             generator: A generator that yields Fibonacci numbers.
21     """
22     a, b = 0, 1
23     for _ in range(n):
24         yield a
25         a, b = b, a + b
26
27 n = int(input("Enter limit: "))
28 for fib_number in fibonacci_generator(n):
29     print(fib_number)
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
6010
6011
6012
6013
6014
6015
6016
6017
6018
6019
6020
6021
6022
6023
6024
6025
6026
6027
6028
6029
6030
6031
6032
6033
6034
6035
6036
6037
6038
6039
60310
60311
60312
60313
60314
60315
60316
60317
60318
60319
60320
60321
60322
60323
60324
60325
60326
60327
60328
60329
60330
60331
60332
60333
60334
60335
60336
60337
60338
60339
603310
603311
603312
603313
603314
603315
603316
603317
603318
603319
603320
603321
603322
603323
603324
603325
603326
603327
603328
603329
603330
603331
603332
603333
603334
603335
603336
603337
603338
603339
6033310
6033311
6033312
6033313
6033314
6033315
6033316
6033317
6033318
6033319
6033320
6033321
6033322
6033323
6033324
6033325
6033326
6033327
6033328
6033329
6033330
6033331
6033332
6033333
6033334
6033335
6033336
6033337
6033338
6033339
60333310
60333311
60333312
60333313
60333314
60333315
60333316
60333317
60333318
60333319
60333320
60333321
60333322
60333323
60333324
60333325
60333326
60333327
60333328
60333329
60333330
60333331
60333332
60333333
60333334
60333335
60333336
60333337
60333338
60333339
603333310
603333311
603333312
603333313
603333314
603333315
603333316
603333317
603333318
603333319
603333320
603333321
603333322
603333323
603333324
603333325
603333326
603333327
603333328
603333329
603333330
603333331
603333332
603333333
603333334
603333335
603333336
603333337
603333338
603333339
6033333310
6033333311
6033333312
6033333313
6033333314
6033333315
6033333316
6033333317
6033333318
6033333319
6033333320
6033333321
6033333322
6033333323
6033333324
6033333325
6033333326
6033333327
6033333328
6033333329
6033333330
6033333331
6033333332
6033333333
6033333334
6033333335
6033333336
6033333337
6033333338
6033333339
60333333310
60333333311
60333333312
60333333313
60333333314
60333333315
60333333316
60333333317
60333333318
60333333319
60333333320
60333333321
60333333322
60333333323
60333333324
60333333325
60333333326
60333333327
60333333328
60333333329
60333333330
60333333331
60333333332
60333333333
60333333334
60333333335
60333333336
60333333337
60333333338
60333333339
603333333310
603333333311
603333333312
603333333313
603333333314
603333333315
603333333316
603333333317
603333333318
603333333319
603333333320
603333333321
603333333322
603333333323
603333333324
603333333325
603333333326
603333333327
603333333328
603333333329
603333333330
603333333331
603333333332
603333333333
603333333334
603333333335
603333333336
603333333337
603333333338
603333333339
6033333333310
6033333333311
6033333333312
6033333333313
6033333333314
6033333333315
6033333333316
6033333333317
6033333333318
6033333333319
6033333333320
6033333333321
6033333333322
6033333333323
6033333333324
6033333333325
6033333333326
6033333333327
6033333333328
6033333333329
6033333333330
6033333333331
6033333333332
6033333333333
6033333333334
6033333333335
6033333333336
6033333333337
6033333333338
6033333333339
60333333333310
60333333333311
60333333333312
60333333333313
60333333333314
60333333333315
60333333333316
60333333333317
60333333333318
60333333333319
60333333333320
60333333333321
60333333333322
60333333333323
60333333333324
60333333333325
60333333333326
60333333333327
60333333333328
60333333333329
60333333333330
60333333333331
60333333333332
60333333333333
60333333333334
60333333333335
60333333333336
60333333333337
60333333333338
60333333333339
603333333333310
603333333333311
603333333333312
603333333333313
603333333333314
603333333333315
603333333333316
603333333333317
603333333333318
603333333333319
603333333333320
603333333333321
603333333333322
603333333333323
603333333333324
603333333333325
603333333333326
603333333333327
603333333333328
603333333333329
603333333333330
603333333333331
603333333333332
603333333333333
603333333333334
603333333333335
603333333333336
603333333333337
603333333333338
603333333333339
6033333333333310
6033333333333311
6033333333333312
6033333333333313
6033333333333314
6033333333333315
6033333333333316
6033333333333317
6033333333333318
6033333333333319
6033333333333320
6033333333333321
6033333333333322
6033333333333323
6033333333333324
6033333333333325
6033333333333326
6033333333333327
6033333333333328
6033333333333329
6033333333333330
6033333333333331
6033333333333332
6033333333333333
6033333333333334
6033333333333335
6033333333333336
6033333333333337
6033333333333338
6033333333333339
60333333333333310
60333333333333311
60333333333333312
60333333333333313
60333333333333314
60333333333333315
60333333333333316
60333333333333317
60333333333333318
60333333333333319
60333333333333320
60333333333333321
60333333333333322
60333333333333323
60333333333333324
60333333333333325
60333333333333326
60333333333333327
60333333333333328
60333333333333329
60333333333333330
60333333333333331
60333333333333332
60333333333333333
60333333333333334
60333333333333335
60333333333333336
60333333333333337
60333333333333338
60333333333333339
603333333333333310
603333333333333311
603333333333333312
603333333333333313
603333333333333314
603333333333333315
603333333333333316
603333333333333317
603333333333333318
603333333333333319
603333333333333320
603333333333333321
603333333333333322
603333333333333323
603333333333333324
603333333333333325
603333333333333326
603333333333333327
603333333333333328
603333333333333329
603333333333333330
603333333333333331
603333333333333332
603333333333333333
603333333333333334
603333333333333335
603333333333333336
603333333333333337
603333333333333338
603333333333333339
6033333333333333310
6033333333333333311
6033333333333333312
6033333333333333313
6033333333333333314
6033333333333333315
6033333333333333316
6033333333333333317
6033333333333333318
6033333333333333319
6033333333333333320
6033333333333333321
6033333333333333322
6033333333333333323
6033333333333333324
6033333333333333325
6033333333333333326
6033333333333333327
6033333333333333328
6033333333333333329
6033333333333333330
6033333333333333331
6033333333333333332
6033333333333333333
6033333333333333334
6033333333333333335
6033333333333333336
6033333333333333337
6033333333333333338
6033333333333333339
60333333333333333310
60333333333333333311
60333333333333333312
60333333333333333313
60333333333333333314
60333333333333333315
60333333333333333316
60333333333333333317
60333333333333333318
60333333333333333319
60333333333333333320
60333333333333333321
60333333333333333322
60333333333333333323
60333333333333333324
60333333333333333325
60333333333333333326
60333333333333333327
60333333333333333328
60333333333333333329
60333333333333333330
60333333333333333331
60333333333333333332
60333333333333333333
60333333333333333334
60333333333333333335
60333333333333333336
60333333333333333337
60333333333333333338
60333333333333333339
603333333333333333310
603333333333333333311
603333333333333333312
603333333333333333313
603333333333333333314
603333333333333333315
603333333333333333316
603333333333333333317
603333333333333333318
603333333333333333319
603333333333333333320
603333333333333333321
603333333333333333322
603333333333333333323
603333333333333333324
603333333333333333325
603333333333333333326
603333333333333333327
603333333333333333328
603333333333333333329
603333333333333333330
603333333333333333331
603333333333333333332
603333333333333333333
603333333333333333334
603333333333333333335
603333333333333333336
603333333333333333337
603333333333333333338
603333333333333333339
6033333333333333333310
6033333333333333333311
6033333333333333333312
6033333333333333333313
6033333333333333333314
6033333333333333333315
6033333333333333333316
6033333333333333333317
6033333333333333333318
6033333333333333333319
6033333333333333333320
6033333333333333333321
6033333333333333333322
6033333333333333333323
6033333333333333333324
6033333333333333333325
6033333333333333333326
6033333333333333333327
6033333333333333333328
6033333333333333333329
6033333333333333333330
6033333333333333333331
6033333333333333333332
6033333333333333333333
6033333333333333333334
6033333333333333333335
6033333333333333333336
6033333333333333333337
6033333333333333333338
6033333333333333333339
60333333333333333333310
60333333333333333333311
60333333333333333333312
60333333333333333333313
60333333333333333333314
60333333333333333333315
60333333333333333333316
60333333333333333333317
60333333333333333333318
60333333333333333333319
60333333333333333333320
60333333333333333333321
60333333333333333333322
60333333333333333333323
60333333333333333333324
60333333333333333333325
60333333333333333333326
60333333333333333333327
60333333333333333333328
60333333333333333333329
60333333333333333333330
60333333333333333333331
60333333333333333333332
60333333333333333333333
60333333333333333333334
60333333333333333333335
60333333333333333333336
60333333333333333333337
60333333333333333333338
60333333333333333333339
603333333333333333333310
603333333333333333333311
603333333333333333333312
603333333333333333333313
603333333333333333333314
603333333333333333333315
603333333333333333333316
603333333333333333333317
603333333333333333333318
603333333333333333333319
603333333333333333333320
603333333333333333333321
603333333333333333333322
603333333333333333333323
603333333333333333333324
603333333333333333333325
603333333333333333333326
603333333333333333333327
603333333333333333333328
603333333333333333333329
603333333333333333333330
603333333333333333333331
603333333333333333333332
603333333333333333333333
603333333333333333333334
603333333333333333333335
603333333333333333333336
603333333333333333333337
603333333333333333333338
603333333333333333333339
6033333333333333333333310
6033333333333333333333311
6033333333333333333333312
6033333333333333333333313
6033333333333333333333314
6033333333333333333333315
6033333333333333333333316
6033333333333333333333317
6033333333333333333333318
6033333333333333333333319
6033333333333333333333320
6033333333333333333333321
60333
```

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")/

```

```

1 # twin_prime had version
2 # refactor into s_prime(n) and is_twin_prime(p1, p2) add docstrings and optimize list of twin primes in given range using si
3 def is_prime(n):
4     ...
5     This function checks if a number is prime.
6     Parameters:
7     n (int): The number to check for primality.
8     returns:
9     bool: True if the number is prime, False otherwise.
10 ...
11     if n <= 1:
12         return False
13     for i in range(2, int(n**0.5) + 1):
14         if n % i == 0:
15             return False
16     return True
17 def is_twin_prime(p1, p2):
18     ...
19     This function checks if two numbers are twin primes.
20     Parameters:
21     p1 (int): The first prime number.
22     p2 (int): The second prime number.
23     returns:
24     bool: True if the numbers are twin primes, False otherwise.
25 ...
26     return is_prime(p1) and is_prime(p2) and abs(p1 - p2) == 2
27 def twin_primes_in_range(start, end):
28     ...
29     This function generates a list of twin primes within a given range.
30     Parameters:
31     start (int): The starting number of the range.
32     end (int): The ending number of the range.
33     returns:
34     list: A list of tuples, each containing a pair of twin primes.
35 ...
36     twin_primes = []
37     for num in range(start, end + 1):
38         if is_prime(num) and is_prime(num + 2):
39             twin_primes.append((num, num + 2))
40     return twin_primes
41 start_range = 1
42 end_range = 100
43 twin_prime_pairs = twin_primes_in_range(start_range, end_range)
44 print(f'Twin primes between {start_range} and {end_range}: {twin_prime_pairs}')
45
46
47
48
49
50
51
52
53
54
55
56
57

```

PROBLEMS 0 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

55
56
57
58 -
59 Twin primes between 1 and 100: [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
60 PS C:\Users\ASUS\Desktop\AIAC 2026\lab 13> & C:\Users\ASUS\AppData\Local\Microsoft\WindowsApps\python3.12.exe "C:/Users/ASUS/Desktop/AIAC 2026/lab 13 - 6.py"
61 Twin primes between 1 and 100: [(3, 5), (5, 7), (11, 13), (17, 19), (29, 31), (41, 43), (59, 61), (71, 73)]
62 PS C:\Users\ASUS\Desktop\AIAC 2026>

```

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.

```

examples of test cases.py demo1.py testcasedemo.py ass 5.1 and 6.py ass 7.4.py demo 9.py demo2.py lab 9.1.py #doc string Untitled-1
lab 13.6.py > ...
1 #Refactoring the Chinese Zodiac Program
2 year = int(input("Enter a year: "))
3 if year % 12 == 0:
4     print("Monkey")
5 elif year % 12 == 1:
6     print("Rooster")
7 elif year % 12 == 2:
8     print("Dog")
9 elif year % 12 == 3:
10    print("Pig")
11 elif year % 12 == 4:
12    print("Rat")
13 elif year % 12 == 5:
14    print("Ox")
15 elif year % 12 == 6:
16    print("Tiger")
17 elif year % 12 == 7:
18    print("Rabbit")
19 elif year % 12 == 8:
20    print("Dragon")
21 elif year % 12 == 9:
22    print("Snake")
23 elif year % 12 == 10:
24    print("Horse")
25 elif year % 12 == 11:
26    print("Goat")
27 #refactor the code to improve readability, maintainability, and structure.
28 zodiac_signs = [
29     "Monkey", "Rooster", "Dog", "Pig", "Rat", "Ox",
30     "Tiger", "Rabbit", "Dragon", "Snake", "Horse", "Goat"
31 ]
32 year = int(input("Enter a year: "))
33 sign_index = year % 12
34 print(zodiac_signs[sign_index])
35
36
37

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

Enter a year: Traceback (most recent call last):
  File "<:Users\ASUS\Desktop\AIAC 2026\lab 13 .6.py>", line 2, in <module>
    year = int(input("Enter a year: "))
    ^^^^^^^^^^^^^^^^^^
KeyboardInterrupt
PS C:\Users\ASUS\Desktop\AIAC 2026> C:\Users\ASUS\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/ASUS/Desktop/AIAC 2026/lab 13 .py"
Enter a year: 2024
Dragon
Enter a year: 2025
Snake
PS C:\Users\ASUS\Desktop\AIAC 2026> █

```

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.
 6. Ensure the program handles edge cases (e.g., 0, negative numbers).

```

lab 13 .6.py > is_harshad_number
31 C:\Users\ASUS\Desktop\AIAC 2026\lab 13 .6.py [ctured Version]18
36 --#Refactoring the Harshad (Niven) Number Checker
37 def is_harshad_number(num):
38     """
39         This function takes a number as input and returns True if the number is a Harshad (Niven) number, otherwise returns False.
40         Parameters:
41             num (int): The number to check.
42         Returns:
43             bool: True if the number is a Harshad number, False otherwise.
44     """
45     temp = num
46     sum_digits = 0
47     while temp > 0:
48         digit = temp % 10
49         sum_digits += digit
50         temp //= 10
51     if sum_digits != 0:
52         return num % sum_digits == 0
53     else:
54         return False
55 number = int(input("Enter a number: "))
56 if is_harshad_number(number):
57     print("True")
58 else:
59     print("False")
60
61
62
63
64
65
66
67
68

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

AAAAAA
IndentError: expected an indented block after 'while' statement on line 39
PS C:\Users\ASUS\Desktop\AIAC 2026> & C:\Users\ASUS\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/ASUS/Desktop/AIAC 2026/lab 13 .6.py"
Enter a number: 18
True
PS C:\Users\ASUS\Desktop\AIAC 2026> & C:\Users\ASUS\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/ASUS/Desktop/AIAC 2026/lab 13 .6.py"
Enter a number: 5
True
PS C:\Users\ASUS\Desktop\AIAC 2026> & C:\Users\ASUS\AppData\Local\Microsoft\WindowsApps\python3.12.exe "c:/Users/ASUS/Desktop/AIAC 2026/lab 13 .6.py"
Enter a number: 9
True
PS C:\Users\ASUS\Desktop\AIAC 2026> []

```

Task 12 – Refactoring the Factorial Trailing Zeros Program

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

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

Problem Statement

The current implementation:

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

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

```
# Factorial Trailing Zeros (Unstructured Version)

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

fact = 1

i = 1

while i <= n:

    fact = fact * i

    i = i + 1

count = 0

while fact % 10 == 0:

    count = count + 1

    fact = fact // 10

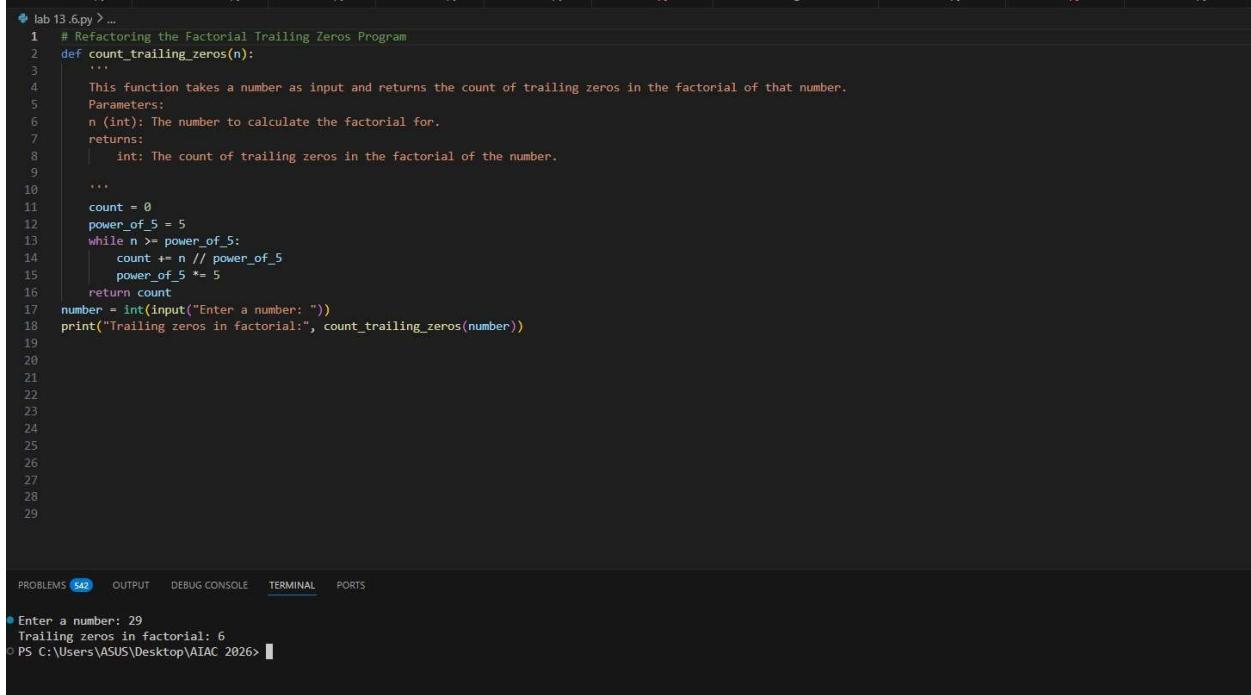
print("Trailing zeros:", count)
```

You must:

1. Create a reusable function: `count_trailing_zeros(n)`
2. The function must:
 - o Accept a non-negative integer n .
 - o Return the number of trailing zeros in $n!$.
3. Do NOT compute the full factorial.

4. Use an optimized mathematical approach (count multiples of 5).
5. Add proper docstrings.
6. Separate user interaction from core logic.
7. Handle edge cases (e.g., negative numbers, zero).

Test Cases Design



```

lab 13.6.py > ...
1  # Refactoring the Factorial Trailing Zeros Program
2 def count_trailing_zeros(n):
3     ...
4     This function takes a number as input and returns the count of trailing zeros in the factorial of that number.
5     Parameters:
6     n (int): The number to calculate the factorial for.
7     Returns:
8         int: The count of trailing zeros in the factorial of the number.
9
10    ...
11    count = 0
12    power_of_5 = 5
13    while n >= power_of_5:
14        count += n // power_of_5
15        power_of_5 *= 5
16    return count
17 number = int(input("Enter a number: "))
18 print("Trailing zeros in factorial:", count_trailing_zeros(number))
19
20
21
22
23
24
25
26
27
28
29

```

PROBLEMS 542 OUTPUT DEBUG CONSOLE TERMINAL PORTS

● Enter a number: 29
Trailing zeros in factorial: 6
○ PS C:\Users\ASUS\Desktop\AIAC 2026>

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 \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]

```

64
65 #write a function collatz sequence that generates the collatz sequence until reaching 1 correctness with pytest
66 def collatz(n):
67     if n <= 0:
68         raise ValueError("Input must be a positive integer.")
69
70     sequence = []
71     while n != 1:
72         sequence.append(n)
73         if n % 2 == 0:
74             n = n // 2
75         else:
76             n = 3 * n + 1
77     sequence.append(1) # Append the last element, which is 1
78     return sequence
79
80 # Test cases
81 print(collatz(6)) # Expected output: [6, 3, 10, 5, 16, 8, 4, 2, 1]
82 print(collatz(1)) # Expected output: [1]
83 print(collatz(3)) # Expected output: [3, 10, 5, 16, 8, 4, 2, 1]

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\HP\OneDrive\Desktop\AI 2026> & C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/HP/OneDrive/Desktop/AI 2026/example of test cases.py"
...
^
SyntaxError: unterminated triple-quoted string literal (detected at line 83)
● PS C:\Users\HP\OneDrive\Desktop\AI 2026> & C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/HP/OneDrive/Desktop/AI 2026/example of test cases.py"
[6, 3, 10, 5, 16, 8, 4, 2, 1]

```

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.

```

64
65 #write a function collatz sequence that generates the collatz sequence until reaching 1 correctness with pytest
66 def collatz(n):
67     if n <= 0:
68         raise ValueError("Input must be a positive integer.")
69
70     sequence = []
71     while n != 1:
72         sequence.append(n)
73         if n % 2 == 0:
74             n = n // 2
75         else:
76             n = 3 * n + 1
77     sequence.append(1) # Append the last element, which is 1
78     return sequence
79
80 # Test cases
81 print(collatz(6)) # Expected output: [6, 3, 10, 5, 16, 8, 4, 2, 1]
82 print(collatz(1)) # Expected output: [1]
83 print(collatz(3)) # Expected output: [3, 10, 5, 16, 8, 4, 2, 1]

```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\HP\OneDrive\Desktop\AI 2026> & C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/HP/OneDrive/Desktop/AI 2026/example of test cases.py"
...
^
SyntaxError: unterminated triple-quoted string literal (detected at line 83)
● PS C:\Users\HP\OneDrive\Desktop\AI 2026> & C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/HP/OneDrive/Desktop/AI 2026/example of test cases.py"
[6, 3, 10, 5, 16, 8, 4, 2, 1]

```

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.

The screenshot shows a Python script named `example of test cases.py` in a code editor. The script defines a function `count_vowels_and_consonants` that takes a string `text` and returns the count of vowels and consonants. It uses a dictionary `vowels` containing lowercase vowels and a set `vowels` for uppercase vowels. The script then iterates through each character in the input string, checks if it's a letter using `char.isalpha()`, and increments the respective count in the `vowel_count` or `consonant_count` variables. Finally, it prints the counts for four different test cases: "Hello World", "Python Programming", "AI 2026", and "OpenAI".

```
23
24     #Write a function vowel & consonant counter vowel and consonant in a given string and return the count of each. Then, write at least 4 normal test cases to verify t
25 def count_vowels_and_consonants(text):
26     vowels = 'aeiouAEIOU'
27     vowel_count = 0
28     consonant_count = 0
29
30     for char in text:
31         if char.isalpha():
32             if char in vowels:
33                 vowel_count += 1
34             else:
35                 consonant_count += 1
36
37     return vowel_count, consonant_count
38
39 # Test cases
40 print(count_vowels_and_consonants("Hello World")) # Expected output: (3
41 print(count_vowels_and_consonants("Python Programming")) # Expected output: (4, 13)
42 print(count_vowels_and_consonants("AI 2026")) # Expected output: (2, 0)
43 print(count_vowels_and_consonants("OpenAI")) # Expected output: (4, 2)
```

The terminal below the code editor shows the execution of the script and its output:

```
PS C:\Users\HP\OneDrive\Desktop\AI 2026> & C:\Users\HP\AppData\Local\Microsoft\WindowsApps\python3.13.exe "c:/Users/HP/OneDrive/Desktop/AI 2026/example of test cases.py"
(3, 7)
(4, 13)
(2, 0)
(4, 2)
PS C:\Users\HP\OneDrive\Desktop\AI 2026>
```