

ASSIGNMENT-9.4

SYED NAZIYA PARVEEN

2303A52394

Batch 42

Task 1: Auto-Generating Function Documentation in a Shared

Codebase

PROMPT: Generate python code to create a simple calculator that can perform basic arithmetic operations such as addition, subtraction, multiplication, and division. The calculator should take user input for the operation and the numbers to be calculated.

CODE:

```
Lab-9.py > ...
1  # generate python code to create a simple calculator that can perform basic arithmetic operations such as add, subtract, multiply, divide
2  def calculator():
3      print("Simple Calculator")
4      print("Select operation:")
5      print("1. Addition (+)")
6      print("2. Subtraction (-)")
7      print("3. Multiplication (*)")
8      print("4. Division (/)")
9      choice = input("Enter choice (1/2/3/4): ")
10     if choice not in ['1', '2', '3', '4']:
11         print("Invalid choice")
12         return
13     num1 = float(input("Enter first number: "))
14     num2 = float(input("Enter second number: "))
15     if choice == '1':
16         result = num1 + num2
17         print(f"{num1} + {num2} = {result}")
18     elif choice == '2':
19         result = num1 - num2
20         print(f"{num1} - {num2} = {result}")
21     elif choice == '3':
22         result = num1 * num2
23         print(f"{num1} * {num2} = {result}")
24     elif choice == '4':
25         if num2 == 0:
26             print("Error: Division by zero is not allowed.")
27         else:
28             result = num1 / num2
29             print(f"{num1} / {num2} = {result}")
30     # run the calculator
31     calculator()
```

OUTPUT:

```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Simple Calculator
Select operation:
1. Addition (+)
2. Subtraction (-)
3. Multiplication (*)
4. Division (/)
Enter choice (1/2/3/4): 1
Enter first number: 5
Enter second number: 3
5.0 + 3.0 = 8.0
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

DOCSTRING PROMPT: generate a calculator using functions for basic operations and generate a google style docstring for each function.each docstring should include a brief description about the function,parametres,return value and an example of how to use the function.

CODE:

```
Lab-09 > calculator

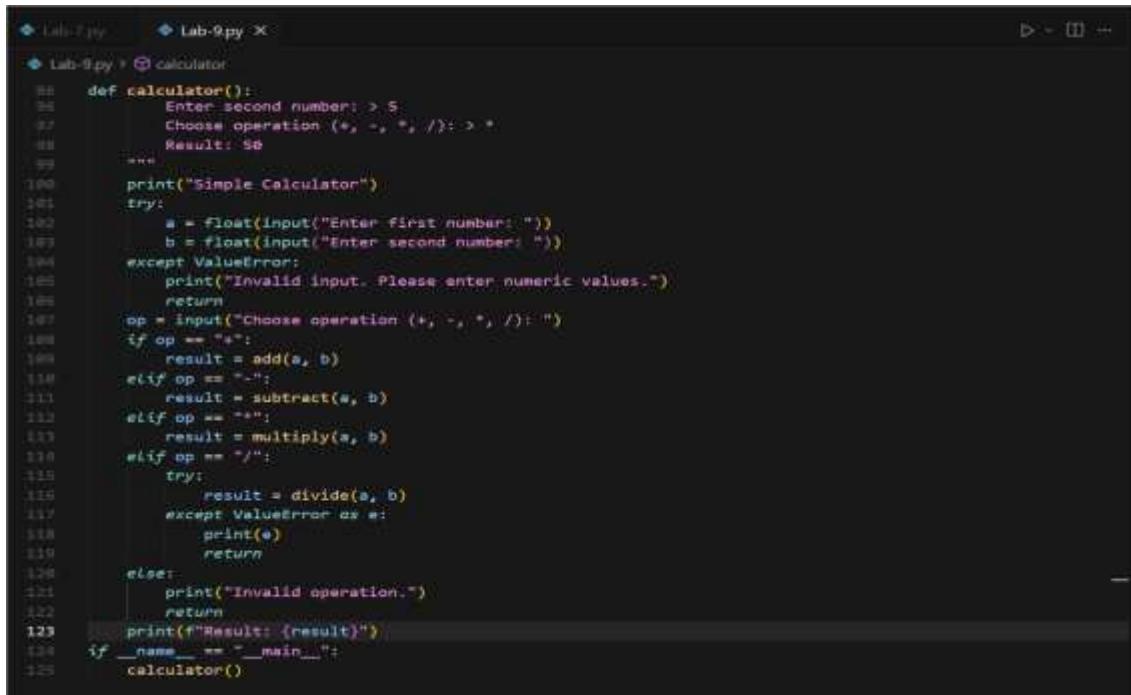
33     generate a calculator using functions for basic operations and generate a google style docstring for each
34
35     def add(a: float, b: float) -> float:
36         """Add two numbers and return their sum.
37
38         Args:
39             a (float): The first addend.
40             b (float): The second addend.
41
42         Returns:
43             float: The sum of "a" and "b".
44
45         Example:
46             >>> add(3, 5)
47             8
48
49         """
50
51     return a + b
52
53     def subtract(a: float, b: float) -> float:
54         """Subtract the second number from the first and return the result.
55
56         Args:
57             a (float): The number to subtract from (minuend).
58             b (float): The number to subtract (subtrahend).
59
60         Returns:
61             float: The result of "a - b".
62
63         Example:
64             >>> subtract(10, 4)
65             6
66
67         """
68
69     return a - b
70
71     def multiply(a: float, b: float) -> float:
72         """Multiply two numbers and return their product.
73
74         Args:
75             a (float): The first factor.
76             b (float): The second factor.
77
78         Returns:
79             float: The product of "a" and "b".
80
81         Example:
```

```
Lab-7.py Lab-9.py x
Lab-9.py > calculator

def multiply(a: float, b: float) -> float:
    """Multiply two numbers and return the product.
    Args:
        a (float): The first number to be multiplied.
        b (float): The second number to be multiplied.
    Returns:
        float: The result of "a * b".
    Raises:
        ValueError: If either "a" or "b" is not a float.
    Example:
        >>> multiply(3, 4)
        12
    """
    return a * b

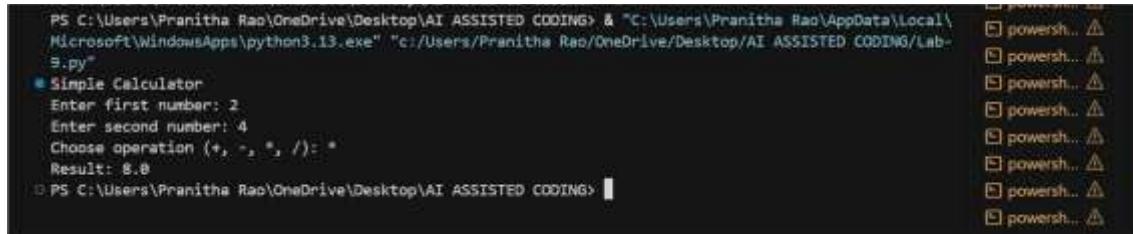
def divide(a: float, b: float) -> float:
    """Divide the first number by the second and return the quotient.
    Args:
        a (float): The dividend (number to be divided).
        b (float): The divisor (number to divide by). Must not be zero.
    Returns:
        float: The result of "a / b".
    Raises:
        ValueError: If "b" is zero.
    Example:
        >>> divide(10, 2)
        5.0
    """
    if b == 0:
        raise ValueError("Cannot divide by zero.")
    return a / b

def calculator():
    """Run a simple text-based calculator for basic arithmetic operations.
    The user is prompted to enter two numbers and an operation choice.
    Supported operations are addition, subtraction, multiplication, and division.
    The function prints the result of the chosen operation.
    Example:
        Example interactive session (user input shown after '>'):
        >>> # This is illustrative; run 'calculator()' in a script/terminal.
        >>> calculator()
        Enter first number: > 10
        Enter second number: > 5
        Choose operation (+, -, *, /): > *
    """
    pass
```



```
Lab-9.py Lab-9.py X
◆ Lab-9.py > calculator
def calculator():
    Enter second number: > 5
    Choose operation (+, -, *, /): > *
    Result: 50
    ...
100 print("Simple Calculator")
101 try:
102     a = float(input("Enter first number: "))
103     b = float(input("Enter second number: "))
104 except ValueError:
105     print("Invalid input. Please enter numeric values.")
106     return
107 op = input("Choose operation (+, -, *, /): ")
108 if op == "+":
109     result = add(a, b)
110 elif op == "-":
111     result = subtract(a, b)
112 elif op == "*":
113     result = multiply(a, b)
114 elif op == "/":
115     try:
116         result = divide(a, b)
117     except ValueError as e:
118         print(e)
119         return
120 else:
121     print("Invalid operation.")
122     return
123 print(f"Result: {result}")
124 if __name__ == "__main__":
125     calculator()
```

OUTPUT:



```
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Simple Calculator
Enter first number: 2
Enter second number: 4
Choose operation (+, -, *, /): *
Result: 8.0
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

JUSTIFICATION: The shared codebase contained several utility functions without proper documentation, making it difficult for new team members to understand their purpose and usage. To solve this problem, an AI-assisted coding tool was used to automatically generate Google-style docstrings for each function. These docstrings clearly describe the function behavior, parameters with data types, return values, and example usage. This approach improved code readability, reduced the learning curve for new developers, and enhanced the overall maintainability and usability of the codebase.

Task 2: Enhancing Readability Through AI-Generated Inline Comments

PROMPT: Write a Python program to search for a number in a list using loops and conditional statements.

CODE:

```

❸ Lab-9.py > ...
185     Write a Python program to search for a number in a list using loops and conditional statements.
186     # Sample list
187     numbers = [10, 20, 30, 40, 50]
188     # Take input from user
189     target = int(input("Enter a number to search: "))
190     found = False    # flag to check if number is found
191     # Loop through the list
192     for num in numbers:
193         if num == target:
194             found = True
195             break
196     # Print result
197     if found:
198         print(target, "is present in the list.")
199     else:
200         print(target, "is NOT present in the list.")

```

OUTPUT:

```

Problems Output Debug Console Terminal Ports + - ^ x
❸ PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Enter a number to search: 30
30 is present in the list.
❸ PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Enter a number to search: 22
22 is NOT present in the list.
❸ PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>

```

DOCSTRING PROMPT: Write a Python program that searches for a value using loops and if conditions. Add a Google-style docstring to explain what the program does, its inputs, and its output.

CODE:

```

❸ Lab-9.py > ...
123     Write a Python program that searches for a value using loops and if conditions. Add a Google-style docstring
124     def search_value(numbers, target):
125         """Search for a target value in a list of numbers using a loop.
126         This function goes through each number in the list and checks
127         (using an if condition) whether it is equal to the target value.
128         If it finds the target, it returns the index at which it was found.
129         If it does not find the target, it returns -1.
130
131         Args:
132             numbers (list[int]): List of integers to search through.
133             target (int): The value to search for in the list.
134
135         Returns:
136             int: The index of the target value (if found); -1 otherwise.
137
138         for index in range(len(numbers)):
139             if numbers[index] == target:
140                 return index
141
142     return -1
143
144     # Example usage
145     if __name__ == "__main__":
146         nums = [10, 20, 30, 40, 50]
147         value_to_find = int(input("Enter a value to search for: "))
148         result_index = search_value(nums, value_to_find)
149         if result_index != -1:
150             print(f"Value {value_to_find} found at index {result_index}.")
151         else:
152             print(f"Value {value_to_find} not found in the list.")

```

OUTPUT:

```

● PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Enter a value to search for: 10
Value 10 found at index 0.
● PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "c:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Enter a value to search for: 11
Value 11 not found in the list.
○ PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> []

```

JUSTIFICATION: This program demonstrates how AI can improve code readability by adding meaningful inline comments and a clear docstring without cluttering the code. In the first part, inline comments explain only the important logic, such as using a flag and breaking the loop once the target is found, making the searching process easier to understand for future maintainers. In the second part, a Google-style docstring is used to clearly describe the purpose of the function, its inputs, and its return value. Overall, the program uses simple loops and conditional statements to search for a number in a list, while AI-generated comments focus on why the logic is used, helping others understand, debug, and extend the code more easily.

Task 3: Generating Module-Level Documentation for a Python Package

PROMPT: Write a simple Python program for a banking system. The program should allow the user to create an account, deposit money, withdraw money, check the balance, and view transaction history. Take input from the user and keep the code minimal and easy to understand.

CODE:

```

● Lab-9 C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\lab-7.py
152 # Write a simple Python program for a banking system. The program should allow the user to create an account,
153 # a Simple Banking System
154 balance = 0.0
155 transactions = [] # Each item: (type, amount, new_balance)
156 account_created = False
157 def create_account():
158     global account_created, balance, transactions
159     if account_created:
160         print("Account already exists.")
161         return
162     name = input("Enter your name: ")
163     print(f"Account created for {name}.")
164     balance = 0.0
165     transactions = []
166     account_created = True
167 def deposit():
168     global balance
169     if not account_created:
170         print("Create an account first.")
171         return
172     try:
173         amount = float(input("Enter amount to deposit: "))
174         if amount <= 0:
175             print("Amount must be positive.")
176             return
177         balance += amount
178         transactions.append(("Deposit", amount, balance))
179         print(f"Deposited {amount}. New balance: {balance}")
180     except ValueError:
181         print("Invalid amount.")
182 def withdraw():
183     global balance
184     if not account_created:

```

```

◆ Lab-7.py  ◆ Lab-9.py X
◆ Lab-9. C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-7.py
183 def withdraw():
184     print("Create an account first.")
185     return
186 try:
187     amount = float(input("Enter amount to withdraw: "))
188     if amount < 0:
189         print("Amount must be positive.")
190         return
191     if amount > balance:
192         print("Insufficient funds.")
193         return
194     balance -= amount
195     transactions.append(("withdraw", amount, balance))
196     print(f"Withdraw {amount}. New Balance: {balance}")
197 except ValueError:
198     print("Invalid amount.")
199 def check_balance():
200     if not account_created:
201         print("Create an account first.")
202         return
203     print(f"Current balance: {balance}")
204 def view_transactions():
205     if not account_created:
206         print("Create an account first.")
207         return
208     if not transactions:
209         print("No transactions yet.")
210         return
211     print("Transaction History:")
212     for i, (t_type, amount, bal_after) in enumerate(transactions, start=1):
213         print(f"{i}: {t_type} ({amount}), Balance after: {bal_after}")
214 def show_menu():
215     print("\n--- Simple Banking System ---")
216
217 def show_menu():
218     print("\n--- Simple Banking System ---")
219     print("1. Create Account")
220     print("2. Deposit Money")
221     print("3. Withdraw Money")
222     print("4. Check Balance")
223     print("5. View Transaction History")
224     print("6. Exit")
225     while True:
226         show_menu()
227         choice = input("Enter your choice (1-6): ")
228         if choice == "1":
229             create_account()
230         elif choice == "2":
231             deposit()
232         elif choice == "3":
233             withdraw()
234         elif choice == "4":
235             check_balance()
236         elif choice == "5":
237             view_transactions()
238         elif choice == "6":
239             print("Thank you for using the banking system! Goodbye!")
240             break
241         else:
242             print("Invalid choice. Please enter a number from 1 to 6.")

```

OUTPUT:

```

Problems  Output  Debug Console  Terminal  Ports
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.10.exe" "C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING\Lab-9.py"
--- Simple Banking System ---
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 1
Enter your name: Pranitha
Account created for Pranitha.

--- Simple Banking System ---
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 2
Enter amount to deposit: 1000
Deposited 1000.0. New balance: 1000.0

--- Simple Banking System ---
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 3
Enter amount to withdraw: 500
Withdraw 500.0. New balance: 500.0

```

```
-- Simple Banking System --
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 4
Current balance: 500.0

-- Simple Banking System --
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 5
Transaction History:
1. Deposit: 1000.0, Balance after: 1000.0
2. Withdraw: 500.0, Balance after: 500.0

-- Simple Banking System --
1. Create Account
2. Deposit Money
3. Withdraw Money
4. Check Balance
5. View Transaction History
6. Exit
Enter your choice (1-6): 6
Thank you for using the banking system. Goodbye!
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

JUSTIFICATION: This program implements a simple banking system using Python functions and user input. It allows a user to create an account and then perform basic banking operations such as depositing money, withdrawing money, checking the current balance, and viewing transaction history. The program uses global variables to store the account balance and a list of transactions, and each function first checks whether an account has been created before performing any operation. A menu-driven loop lets the user choose actions repeatedly until they decide to exit. Overall, the code is minimal, easy to understand, and clearly shows how functions, conditionals, and loops work together in a real-world example.

Task 4: Converting Developer Comments into Structured Docstrings

PROMPT: Create a basic Python program to manage student results. Accept marks for multiple exams from the user and use separate functions to find the total, average, grade, highest score, lowest score, and show the final report. Add detailed comments to explain the logic and keep the program simple.

CODE:

```
◆ Lab-7.py ◆ Lab-8.py X
└ CAUsers\Pranitha Rao\OneDrive\Desktop\W ASSISTED CODING\Lab-7.py
  264 #Create a basic Python program to manage student results. Accept marks for multiple exams from the user and calculate the total, average, grade, highest and lowest marks.
  265 def calculate_total(marks):
  266     """
  267     Calculate the total of all exam marks.
  268     Args:
  269         marks (list): List of marks obtained in different exams
  270     Returns:
  271         float: Sum of all marks
  272     """
  273     # Initialize total to zero
  274     total = 0
  275     # Loop through each mark and add it to the total
  276     for mark in marks:
  277         total += mark
  278     return total
  279 def calculate_average(marks):
  280     """
  281     Calculate the average of all exam marks.
  282     Args:
  283         marks (list): List of marks obtained in different exams
  284     Returns:
  285         float: Average of all marks (rounded to 2 decimal places)
  286     """
  287     # Check if the list is empty to avoid division by zero
  288     if len(marks) == 0:
  289         return 0.0
  290     # Calculate average using the calculate_total function
  291     total = calculate_total(marks)
  292     # Calculate average by dividing total by number of exams
  293     average = total / len(marks)
  294     # Round to 2 decimal places for better readability
  295     return round(average, 2)
  296 def calculate_grade(average):
  297
```

```
◆ Lab-7.py ◆ Lab-8.py X
└ CAUsers\Pranitha Rao\OneDrive\Desktop\W ASSISTED CODING\Lab-7.py
  298     """
  299     Determine the grade based on the average marks.
  300     Args:
  301         average (float): Average marks obtained
  302     Returns:
  303         str: Grade (A, B, C, D, or F)
  304     """
  305     # Use if-elif conditions to assign grades based on average marks
  306     if average >= 90:
  307         return "A"
  308     elif average >= 80:
  309         return "B"
  310     elif average >= 70:
  311         return "C"
  312     elif average >= 60:
  313         return "D"
  314     else:
  315         return "F"
  316 def find_highest_score(marks):
  317     """
  318     Find the highest score among all exam marks.
  319     Args:
  320         marks (list): List of marks obtained in different exams
  321     Returns:
  322         float: Highest mark in the list
  323     """
  324     # Check if list is empty
  325     if len(marks) == 0:
  326         return 0.0
  327     # Initialize highest with the first mark
  328     highest = marks[0]
```

```
◆ Lab-7.py ◆ Lab-9.py X
└ CAUsers\Pranitha Rao\OneDrive\Desktop\W ASSISTED CODING\Lab-7.py
  329     def find_highest_score(marks):
  330         """
  331         Loop through remaining marks and update highest if a larger value is found
  332         for mark in marks[1:]:
  333             if mark > highest:
  334                 highest = mark
  335         return highest
  336     def find_lowest_score(marks):
  337         """
  338         Find the lowest score among all exam marks.
  339         Args:
  340             marks (list): List of marks obtained in different exams
  341         Returns:
  342             float: Lowest mark in the list
  343         """
  344         # Check if list is empty
  345         if len(marks) == 0:
  346             return 0.0
  347         # Initialize lowest with the first mark
  348         lowest = marks[0]
  349         # Loop through remaining marks and update lowest if a smaller value is found
  350         for mark in marks[1:]:
  351             if mark < lowest:
  352                 lowest = mark
  353         return lowest
  354     def show_final_report(marks, student_name):
  355         """
  356         Display a comprehensive report of student results.
  357         Args:
  358             marks (list): List of marks obtained in different exams
  359             student_name (str): Name of the student
  360         """
  361         # Calculate all required statistics using the respective functions
  362         total = calculate_total(marks)
```

```
◆ Lab-7.py ◆ Lab-9.py X
◆ Lab-9.py > main
323     def show_final_report(marks, student_name):
324         total = calculate_total(marks)
325         average = calculate_average(marks)
326         grade = calculate_grade(average)
327         highest = find_highest_score(marks)
328         lowest = find_lowest_score(marks)
329         # Display the formatted report
330         print("\n" + "="*50)
331         print("STUDENT RESULTS REPORT")
332         print("="*50)
333         print(f"Student Name: {student_name}")
334         print(f"Number of Exams: {len(marks)}")
335         print(f"\nEnter Marks:")
336         # Display each exam mark with its number
337         for i, mark in enumerate(marks, start=1):
338             print(f" Exam {i}: {mark}")
339         print(f"\nTotal Marks: {total}")
340         print(f"Average Marks: {average}")
341         print(f"Grade: {grade}")
342         print(f"\nHighest Score: {highest}")
343         print(f"\nLowest Score: {lowest}")
344         print("="*50 + "\n")
345     def main():
346         """
347             Main function to run the student results management program.
348             Handles user input and coordinates all functions.
349         """
350         # Get student name from user
351         student_name = input("Enter student name: ")
352         # Get number of exams from user
353         try:
354             num_exams = int(input("Enter the number of exams: "))
355             # Validate that number of exams is positive
356         except ValueError:
357             print("Invalid input. Please enter a valid number.")
358             return
359         # Initialize empty list to store marks
360         marks = []
361         # Loop to get marks for each exam
362         print("\nEnter marks for {num_exams}-exam(s):")
363         for i in range(1, num_exams + 1):
364             while True:
365                 try:
366                     # Get mark for each exam
367                     mark = float(input(f"Enter marks for Exam {i}: "))
368                     # Validate that marks are non-negative
369                     if mark < 0:
370                         print("Marks cannot be negative. Please enter again.")
371                         continue
372                     # Add valid mark to the list
373                     marks.append(mark)
374                     break
375                 except ValueError:
376                     print("Invalid input. Please enter a valid number.")
377             # Display the final report using the show_final_report function
378             show_final_report(marks, student_name)
379         # Run the program when the script is executed
380         if __name__ == "__main__":
381             main()
```

```
◆ Lab-7.py ◆ Lab-9.py X
◆ Lab-9.py > main
362     def main():
363         try:
364             num_exams = int(input("Enter the number of exams: "))
365             # Validate that number of exams is positive
366             if num_exams <= 0:
367                 print("Number of exams must be greater than 0.")
368                 return
369         except ValueError:
370             print("Invalid input. Please enter a valid number.")
371             return
372         # Initialize empty list to store marks
373         marks = []
374         # Loop to get marks for each exam
375         print("\nEnter marks for {num_exams}-exam(s):")
376         for i in range(1, num_exams + 1):
377             while True:
378                 try:
379                     # Get mark for each exam
380                     mark = float(input(f"Enter marks for Exam {i}: "))
381                     # Validate that marks are non-negative
382                     if mark < 0:
383                         print("Marks cannot be negative. Please enter again.")
384                         continue
385                     # Add valid mark to the list
386                     marks.append(mark)
387                     break
388                 except ValueError:
389                     print("Invalid input. Please enter a valid number.")
390             # Display the final report using the show_final_report function
391             show_final_report(marks, student_name)
392         # Run the program when the script is executed
393         if __name__ == "__main__":
394             main()
```

DOCSTRING PROMPT: Write a Python program for a student grade system that takes marks from the user and uses separate functions to calculate total marks, average, grade, highest and lowest scores, and display a final report. Convert all explanatory comments into clear Google-style or NumPy-style docstrings, remove unnecessary inline comments, include a proper module-level docstring with an example, and keep the code minimal with standardized documentation.

CODE:

```
◆ Lab-7.py ④ Lab-8.py X
◆ Lab-8.py + ④ main
327     """Write a Python program for a student grade system that takes marks from the user and uses separate function
328     """
329     """Student Grade System.
330     A program to manage student grades by accepting marks from the user and
331     calculating various statistics including total, average, grade, highest,
332     and lowest scores.
333     Example:
334         >>> if __name__ == "__main__":
335             main()
336
337         Enter student name: John Doe
338         Enter the number of exams: 5
339         Enter marks for 5 exams:
340             Enter Marks for Exam 1: 85
341             Enter Marks for Exam 2: 92
342             Enter Marks for Exam 3: 78
343             Enter Marks for Exam 4: 98
344             Enter Marks for Exam 5: 88
345
346         STUDENT RESULT REPORT
347         -----
348         Student Name: John Doe
349         Number of Exams: 5
350
351         Total Marks: 433
352         Average Marks: 86.6
353         Grade: B
354         Highest Score: 98
355         Lowest Score: 78
356
357     def calculate_total(marks):
358         """Calculate the sum of all marks.
359         Args:
360             marks (list[float]): List of exam marks.
361
362         Returns:
363             float: Sum of all marks in the list.
364
365     return sum(marks)
366
367     def calculate_average(marks):
368         """Calculate the average of all marks.
369         Args:
370             marks (list[float]): List of exam marks.
371         Returns:
372             float: Average of marks rounded to 2 decimal places. Returns 0.0
373             if the list is empty.
374
375         if len(marks) == 0:
376             return 0.0
377         return round(calculate_total(marks) / len(marks), 2)
378
379     def calculate_grade(average):
380         """Determine the letter grade based on average marks.
381         Args:
382             average (float): Average marks of the student.
383         Returns:
384             str: Letter grade ('A', 'B', 'C', 'D', or 'F') based on the
385                 following scale:
386                 - A: 90 and above
387                 - B: 80-89
388                 - C: 70-79
389                 - D: 60-69
390                 - F: Below 60
391
392         if average >= 90:
393             return "A"
394         elif average >= 80:
395             return "B"
396
397     def calculate_grade(average):
398         """Determine the letter grade based on average marks.
399         Args:
400             average (float): Average marks of the student.
401         Returns:
402             str: Letter grade ('A', 'B', 'C', 'D', or 'F') based on the
403                 following scale:
404                 - A: 90 and above
405                 - B: 80-89
406                 - C: 70-79
407                 - D: 60-69
408                 - F: Below 60
409
410         if average >= 90:
411             return "A"
412         elif average >= 80:
413             return "B"
414         elif average >= 70:
415             return "C"
416         elif average >= 60:
417             return "D"
418         else:
419             return "F"
420
421     def find_highest_score(marks):
422         """Find the highest score from the list of marks.
423         Args:
424             marks (list[float]): List of exam marks.
425         Returns:
426             float: Highest mark in the list. Returns 0.0 if the list is empty.
427
428         if len(marks) == 0:
429             return 0.0
430         return max(marks)
431
432     def find_lowest_score(marks):
433         """Find the lowest score from the list of marks.
434         Args:
435             marks (list[float]): List of exam marks.
436         Returns:
437             float: Lowest mark in the list. Returns 0.0 if the list is empty.
438
439         if len(marks) == 0:
440             return 0.0
441         return min(marks)
442
443     def display_final_report(marks, student_name):
444         """Display a formatted report with all student statistics.
445         Args:
446             marks (list[float]): List of exam marks.
447             student_name (str): Name of the student.
448
449         Returns:
450             None
451
452         print("STUDENT RESULT REPORT")
453         print("-----")
454         print(f"Student Name: {student_name}")
455         print(f"Number of Exams: {len(marks)}")
456
457         total_marks = calculate_total(marks)
458         average_marks = calculate_average(marks)
459         grade = calculate_grade(average_marks)
460
461         print(f"Total Marks: {total_marks}")
462         print(f"Average Marks: {average_marks:.2f}")
463         print(f"Grade: {grade}")
464         print(f"Highest Score: {max(marks)}")
465         print(f"Lowest Score: {min(marks)}")
```

```

  ● Lab-9.py   ● Lab-9.py X
  ● Lab-9.py + main
  424     def display_final_report(marks, student_name):
  425         """ This Function prints the report to stdout.
  426
  427         total = calculate_total(marks)
  428         average = calculate_average(marks)
  429         grade = calculate_grade(average)
  430         highest = find_highest_score(marks)
  431         lowest = find_lowest_score(marks)
  432         print("\n" + "=" * 50)
  433         print("STUDENT RESULTS REPORT")
  434         print("=" * 50)
  435         print(f"Student Name: {student_name}")
  436         print(f"Number of Exams: {len(marks)}")
  437         print(f"\nExam Marks:")
  438         for i, mark in enumerate(marks, start=1):
  439             print(f"    Exam {i}: {mark}")
  440         print(f"\nTotal Marks: {total}")
  441         print(f"Average Marks: {average}")
  442         print(f"Grade: {grade}")
  443         print(f"Highest Score: {highest}")
  444         print(f"Lowest Score: {lowest}")
  445         print("=" * 50)
  446     def get_marks_from_user(num_exams):
  447         """Collect exam Marks from the user via input.
  448
  449         Args:
  450             num_exams (int): Number of exams to collect marks for.
  451
  452         Returns:
  453             list[float]: List of valid exam marks entered by the user.
  454
  455         """
  456         marks = []
  457         print("\nEnter marks for (num_exams) exam(s):")
  458         for i in range(1, num_exams + 1):
  459             while True:
  460
  461     ● Lab-9.py   ● Lab-9.py X
  462     ● Lab-9.py + main
  463     def get_marks_from_user(num_exams):
  464
  465         try:
  466             mark = float(input("Enter marks for Exam (1) = "))
  467             if mark < 0:
  468                 print("Marks cannot be negative. Please enter again.")
  469                 continue
  470             marks.append(mark)
  471             break
  472         except ValueError:
  473             print("Invalid input. Please enter a valid number.")
  474
  475     return marks
  476
  477     def main():
  478         """Main function to run the student grade system.
  479         Prompts user for student name and number of exams, collects marks,
  480         and displays the final report.
  481
  482         """
  483         student_name = input("Enter student name: ")
  484         try:
  485             num_exams = int(input("Enter the number of exams: "))
  486             if num_exams <= 0:
  487                 print("Number of exams must be greater than 0.")
  488                 return
  489             except ValueError:
  490                 print("Invalid input. Please enter a valid number.")
  491
  492             marks = get_marks_from_user(num_exams)
  493             display_final_report(marks, student_name)
  494
  495             if __name__ == "__main__":
  496                 main()

```

OUTPUT:

```

Problems Output Debug Console Terminal Ports
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.11.exe" "C:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
Enter student name: Pranitha
Enter the number of exams: 5

Enter marks for 5 exam(s):
Enter marks for Exam 1: 88
Enter marks for Exam 2: 65
Enter marks for Exam 3: 77
Enter marks for Exam 4: 98
Enter marks for Exam 5: 36

-----
STUDENT RESULTS REPORT
-----
Student Name: Pranitha
Number of Exams: 5

Exam Marks:
Exam 1: 88.0
Exam 2: 65.0
Exam 3: 77.0
Exam 4: 98.0
Exam 5: 36.0

-----
STUDENT RESULTS REPORT
-----
Student Name: Pranitha
Number of Exams: 5

Exam Marks:
Exam 1: 88.0
Exam 2: 65.0
Exam 3: 77.0
Exam 4: 98.0
Exam 5: 36.0

```

```
Exam_Score.py
Total Marks: 356.0
Average Marks: 71.2
Grade: C
Highest Score: 98.0
Lowest Score: 36.0
=====
PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING>
```

JUSTIFICATION: This code shows how long inline developer comments can be converted into clean, standardized Google-style docstrings to improve documentation quality. The student grade system accepts exam marks from the user and uses well-defined functions to calculate total marks, average, grade, highest score, and lowest score, then displays a final report. By moving explanations into structured docstrings, the function bodies become cleaner and easier to read, while the documentation clearly explains each function's purpose, inputs, and outputs. Overall, the code is more consistent, maintainable, and suitable for realworld or shared projects.

Task 5: Building a Mini Automatic Documentation Generator

PROMPT: Generate a python code in a .py file for a To-Do List Manager which includes multiple utility functions like add_task, remove_task, update_task. Take user inputs for the code. Generate a minimal code.

CODE:

```
Lab_8.py
# Generate a python code in a .py file for a To-Do List Manager which includes multiple utility functions like add_task, remove_task, update_task. Take user inputs for the code. Generate a minimal code.

# To-Do List Manager
tasks = []

def add_task():
    """Add a new task to the list."""
    task = input("Enter task to add: ")
    if task.strip():
        tasks.append(task)
        print(f"Task '{task}' added successfully!")
    else:
        print("Task cannot be empty.")

def remove_task():
    """Remove a task from the list."""
    if not tasks:
        print("No tasks to remove.")
        return
    view_tasks()
    try:
        index = int(input("Enter task number to remove: ")) - 1
        if 0 <= index < len(tasks):
            removed = tasks.pop(index)
            print(f"Task '{removed}' removed successfully!")
        else:
            print("Invalid task number.")
    except ValueError:
        print("Invalid input. Please enter a number.")

def update_task():
    """Update an existing task."""
    if not tasks:
        print("No tasks to update.")
        return
    view_tasks()
    print("Enter task number to update: ")
```

```
◆ Lab-7.py ◆ Lab-9.py X
◆ Lab-9.py > ...
468 def update_task():
469     try:
470         index = int(input("Enter task number to update: ")) - 1
471         if 0 <= index < len(tasks):
472             new_task = input("Enter new task description: ")
473             if new_task.strip():
474                 old_task = tasks[index]
475                 tasks[index] = new_task
476                 print(f"Task '{old_task}' updated to '{new_task}'")
477             else:
478                 print("Task cannot be empty.")
479         else:
480             print("Invalid task number.")
481     except ValueError:
482         print("Invalid input. Please enter a number.")
483 def view_tasks():
484     """Display all tasks."""
485     if not tasks:
486         print("No tasks in the list.")
487         return
488     print("\n--- Your Tasks ---")
489     for i, task in enumerate(tasks, start=1):
490         print(f"{i}: {task}")
491     print()
492 def show_menu():
493     """Display the main menu."""
494     print("\n--- To-Do List Manager ---")
495     print("1. Add Task")
496     print("2. Remove Task")
497     print("3. Update Task")
498     print("4. View Tasks")
499     print("5. Exit")
500 def main():
501     """Main program loop."""
502     while True:
503         show_menu()
504         choice = input("Enter your choice (1-5): ")
505         if choice == "1":
506             add_task()
507         elif choice == "2":
508             remove_task()
509         elif choice == "3":
510             update_task()
511         elif choice == "4":
512             view_tasks()
513         elif choice == "5":
514             print("Thank you for using To-Do List Manager. Goodbye!")
515             break
516         else:
517             print("Invalid choice. Please enter a number from 1 to 5.")
518 if __name__ == "__main__":
519     main()
```

```
◆ Lab-7.py ◆ Lab-9.py X
◆ Lab-9.py > ...
497 def main():
498     """Main program loop."""
499     while True:
500         show_menu()
501         choice = input("Enter your choice (1-5): ")
502         if choice == "1":
503             add_task()
504         elif choice == "2":
505             remove_task()
506         elif choice == "3":
507             update_task()
508         elif choice == "4":
509             view_tasks()
510         elif choice == "5":
511             print("Thank you for using To-Do List Manager. Goodbye!")
512             break
513         else:
514             print("Invalid choice. Please enter a number from 1 to 5.")
515 if __name__ == "__main__":
516     main()
```

DOCSTRING PROMPT: Generate the same python code with proper doctrings for a To-Do List Manager which includes multiple utility functions like add_task, remove_task, update_task. Take user inputs for the code. The task is to detect functions and classes and insert placeholder google style docstrings for each detected function or class. The goal is documentation scaffolding. Generate a minimal and efficient code.

CODE:

```
Lab-7.py Lab-9.py X
Lab-9.py
513 demonstrate the same python code with proper docstrings for a To-Do List Manager which includes multiple utility
514
515 To-Do List Manager:
516 This module provides a simple command-line interface for managing a to-do list.
517 Users can add, remove, update, and view tasks through an interactive menu.
518 Example:
519     Run the script to start the interactive to-do list manager!
520     $ python todo_manager.py
521 ...
522
523 class TodoListManager:
524     """A simple to-do list manager class.
525     This class manages a collection of tasks, allowing users to add, remove,
526     update, and view tasks through various utility methods.
527     Attributes:
528         tasks (list[str]): A list of task descriptions stored as strings.
529     Example:
530         >>> manager = TodoListManager()
531         >>> manager.add_task("Buy groceries")
532         Task 'Buy groceries' added successfully!
533
534     def __init__(self):
535         """Initialize an empty TodoListManager.
536         Creates a new instance with an empty task list.
537         """
538         self.tasks = []
539     def add_task(self, task: str) -> bool:
540         """Add a new task to the to-do list.
541         Args:
542             task (str): The task description to add. Must not be empty or
543                         whitespace-only.
544         Returns:
545             bool: True if the task was successfully added, False otherwise.
546     Example:
```

```
Lab-9.py <--> Lab-9.py X
Lab-9.py +
class TodoListManager:
    def add_task(self, task: str) -> bool:
        >>> manager = TodoListManager()
        >>> manager.add_task("Complete assignment")
        True

        if task.strip():
            self.tasks.append(task)
            print(f"Task '{task}' added successfully!")
            return True
        else:
            print("Task cannot be empty.")
            return False

    def remove_task(self, index: int) -> bool:
        >>> remove a task from the to-do list by its index.
        Args:
            Index (int): The 1-based index of the task to remove. The index
                should be between 1 and the number of tasks.
        Returns:
            bool: True if the task was successfully removed, False otherwise.
        Examples:
            >>> manager = TodoListManager()
            >>> manager.add_task("Task 1")
            >>> manager.remove_task(1)
            True

            if not self.tasks:
                print("No tasks to remove.")
                return False
            if 1 <= index < len(self.tasks):
                removed = self.tasks.pop(index - 1)
                print(f"Task '{removed}' removed successfully!")
                return True
```

```
 1 class TodoListManager:
 2     def __init__(self):
 3         self.tasks = []
 4
 5     def add_task(self, task: str) -> bool:
 6         self.tasks.append(task)
 7         return True
 8
 9     def remove_task(self, index: int) -> bool:
10         if index < 0 or index >= len(self.tasks):
11             print("Invalid task number!")
12             return False
13
14     def update_task(self, index: int, new_task: str) -> bool:
15         """Update an existing task with a new description.
16
17         Args:
18             index (int): The 1-based index of the task to update. The index
19             should be between 1 and the number of tasks.
20             new_task (str): The new task description. Must not be empty or
21                             whitespace-only.
22
23         Returns:
24             bool: True if the task was successfully updated, False otherwise.
25
26         Examples:
27             >>> manager = TodoListManager()
28             >>> manager.add_task("Old task")
29             >>> manager.update_task(1, "New task")
30             True
31
32             If not self.tasks:
33                 print("No tasks to update...")
34                 return False
35
36             if index < len(self.tasks):
37                 if new_task.strip():
38                     old_task = self.tasks[index - 1]
39                     self.tasks[index - 1] = new_task
40                     print(f"(old_task) updated to {(new_task)}!")
41                     return True
42                 else:
43                     print("Task cannot be empty!")
44                     return False
45
46             else:
47                 print("Index out of range!")
48                 return False
```

```
Lab-Todo  Lab-B.py X
Lab-B.py
class TodolistManager:
    def update_task(self, index: int, new_task: str) -> bool:
        print("Invalid task number.")
        return False
    def view_tasks(self) -> None:
        """Display all tasks in the to-do list.
        Prints a numbered list of all tasks. If the list is empty, displays
        an appropriate message.
        Returns:
            None
        Example:
            >>> manager = TodolistManager()
            >>> manager.add_task("Task 1")
            >>> manager.add_task("Task 2")
            >>> manager.view_tasks()
            >>> Your Tasks
            1. Task 1
            2. Task 2
        """
        if not self.tasks:
            print("No tasks in the list.")
            return
        print("\n--- Your Tasks ---")
        for i, task in enumerate(self.tasks, start=1):
            print(f"\t{i}. {task}")
    def get_task_count(self) -> int:
        """Get the total number of tasks in the list.
        Returns:
            int: The number of tasks currently in the list.
        Example:
            >>> manager = TodolistManager()
            >>> manager.add_task("Task 1")
```

```
Lab-Todo  Lab-B.py X
Lab-B.py
class TodolistManager:
    def get_task_count(self) -> int:
        >>> manager.get_task_count()
        3
        """
        return len(self.tasks)
    def show_menu() -> None:
        """Display the main menu options for the to-do list manager.
        Prints a formatted menu with numbered options for all available
        operations.
        Returns:
            None
        Example:
            >>> show_menu()
            --- To-Do List Manager ---
            1. Add Task
            2. Remove Task
            3. Update Task
            4. View Tasks
            5. Exit
        """
        print("\n--- To-Do List Manager ---")
        print("1. Add Task")
        print("2. Remove Task")
        print("3. Update Task")
        print("4. View Tasks")
        print("5. Exit")
    def get_user_choice() -> str:
        """Get user's menu choice from standard input.
        Prompts the user to enter a choice and returns it as a string.
        Returns:
            str: The user's input choice.
        Example:
            >>> get_user_choice()
            Enter your choice (1-5): 1
```

```
Lab-Todo  Lab-B.py X
Lab-B.py
def get_user_choice() -> str:
    >>> # User enters "1"
    >>> choice = get_user_choice()
    Enter your choice (1-5): 1
    >>> choice
    '1'
    """
    return input("Enter your choice (1-5): ")
def get_task_input() -> str:
    """Get task description from user input.
    Prompts the user to enter a task description.
    Returns:
        str: The task description entered by the user.
    Example:
        >>> # User enters "Buy groceries"
        >>> task = get_task_input()
        Enter task to add: Buy groceries
        >>> task
        'Buy groceries'
    """
    return input("Enter task to add: ")
def get_task_index() -> int:
    """Get task index from user input.
    Prompts the user to enter a task number and converts it to an integer.
    Handles invalid input by returning -1.
    Returns:
        int: The task index (1-based) if valid input, -1 otherwise.
    Example:
        >>> # User enters "2"
        >>> index = get_task_index()
        Enter task number to remove: 2
        >>> index
        2
```

```

  1 Lab-9.py  Lab-9.py X
  2 Lab-9.py ? ...
  3
  4 def main() -> None:
  5     """ python todo_manager.py
  6
  7     manager = TodoListManager()
  8     while True:
  9         show_menu()
 10         choice = get_user_choice()
 11         if choice == "1":
 12             task = get_task_input()
 13             manager.add_task(task)
 14         elif choice == "2":
 15             if manager.tasks:
 16                 manager.view_tasks()
 17                 index = get_task_index()
 18                 if index != -1:
 19                     manager.remove_task(index)
 20             elif choice == "3":
 21                 if manager.tasks:
 22                     manager.view_tasks()
 23                     index = get_update_index()
 24                     if index != -1:
 25                         new_task = get_new_task_description()
 26                         manager.update_task(index, new_task)
 27             elif choice == "4":
 28                 manager.view_tasks()
 29             elif choice == "5":
 30                 print("Thank you for using To-Do List Manager. Goodbye!")
 31             break
 32         else:
 33             print("Invalid choice. Please enter a number from 1 to 5.")
 34     if __name__ == "__main__":
 35         main()

```

OUTPUT:

```

  PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "<:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
  --- To-Do List Manager ---
  1. Add Task
  2. Remove Task
  3. Update Task
  4. View Tasks
  5. Exit
  Enter your choice (1-5): 1
  Enter task to add: Dance
  Task 'Dance' added successfully!
  --- To-Do List Manager ---
  1. Add Task
  2. Remove Task
  3. Update Task
  4. View Tasks
  5. Exit
  Enter your choice (1-5): 2
  --- Your Tasks ---
  1. Dance

  Enter task number to remove: 3
  Invalid task number.

  --- To-Do List Manager ---
  1. Add Task
  2. Remove Task
  3. Update Task
  4. View Tasks
  5. Exit

```

```

  PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> & "C:\Users\Pranitha Rao\AppData\Local\Microsoft\WindowsApps\python3.13.exe" "<:/Users/Pranitha Rao/OneDrive/Desktop/AI ASSISTED CODING/Lab-9.py"
  --- To-Do List Manager ---
  1. Add Task
  2. Remove Task
  3. Update Task
  4. View Tasks
  5. Exit
  Enter your choice (1-5): 4
  --- Your Tasks ---
  1. Dance

  --- To-Do List Manager ---
  1. Add Task
  2. Remove Task
  3. Update Task
  4. View Tasks
  5. Exit
  Enter your choice (1-5): 5
  Thank you for using To-Do List Manager. Goodbye!
  PS C:\Users\Pranitha Rao\OneDrive\Desktop\AI ASSISTED CODING> []

```

JUSTIFICATION: This example demonstrates how a simple to-do list program can be enhanced by automatically adding structured Google-style docstrings to functions and classes. The script manages tasks using a TodoListManager class and provides features like adding, removing, updating, and viewing tasks through user input. By replacing informal comments with clear docstrings, the code becomes easier to understand, more consistent, and suitable for shared or internal projects. Overall, it shows how AI-assisted documentation scaffolding can quickly improve code readability and maintainability without changing the program's core logic.