Anjali Maripudi

2403A51272

Batch-12

AI ASSISTED CODING – ASSIGNMENT 9.2

**Task Description #1** (Documentation – Google-Style Docstrings without providing any input- output examples. for Python Functions)

Use AI to add Google-style docstrings to all functions in a given Python script.

Prompt –

add Google-style docstrings to all functions in the following Python script. def is\_prime(n):

"""Check if a number is prime.""" if n <= 1:

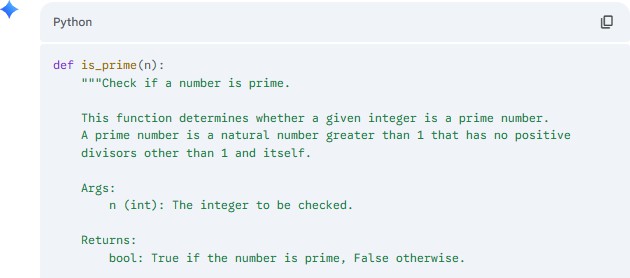
return False

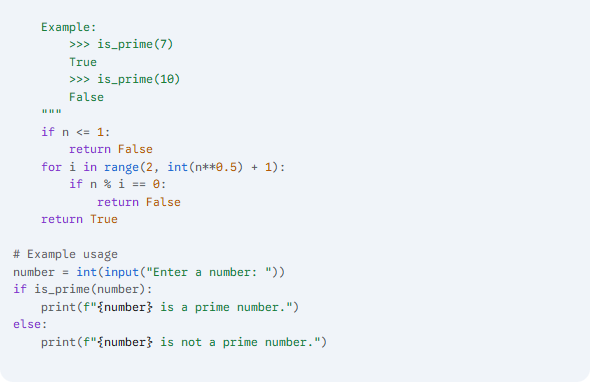
for i in range(2, int(n\*\*0.5) + 1): if n % i == 0:

return False return True

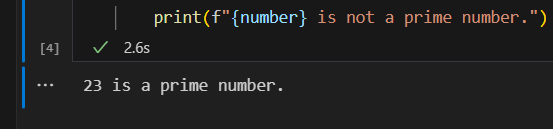
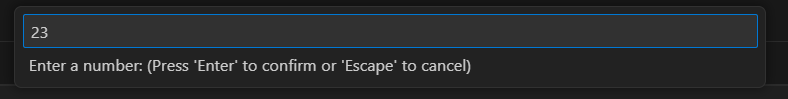
generate docstrings without providing any input-output examples.

Code –





Output –



**Task Description #2** (Documentation – Inline Comments for Complex Logic)

Use AI to add meaningful inline comments to a Python program explaining only complex logic parts.

Prompt –

Code –

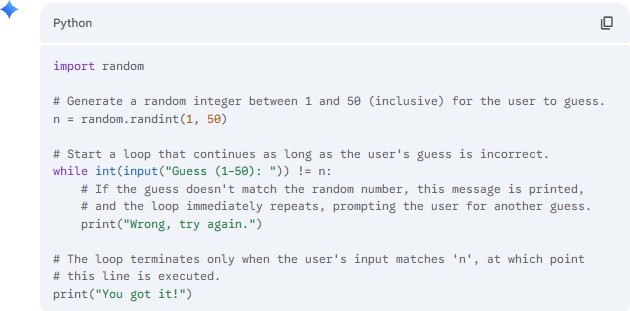
import random

n = random.randint(1, 50)

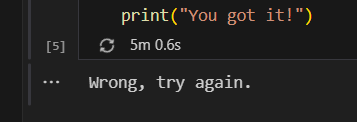
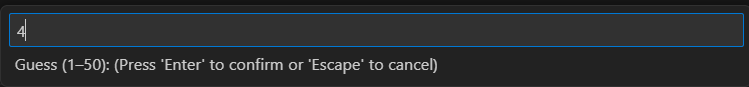
while int(input("Guess (1–50): ")) != n: print("Wrong, try again.")

print("You got it!")

add meaningful inline comments to a Python program explaining only complex logic parts skip obvious syntax explanations and focus only on tricky or non-intuitive code sections.



Output –



**Task Description #3** (Documentation – Module-Level Documentation)

Use AI to create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of a Python file.

Prompt – def greet():

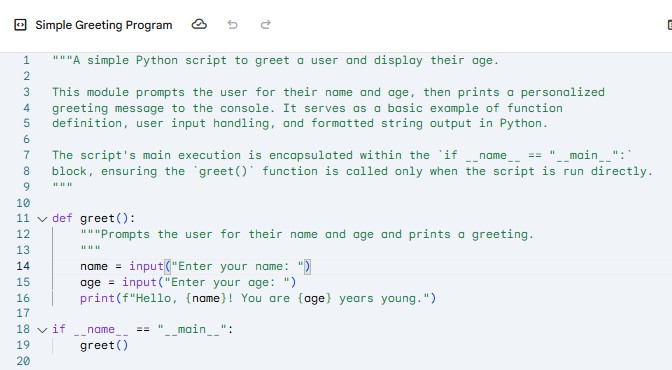
name = input("Enter your name: ") age = input("Enter your age: ")

print(f"Hello, {name}! You are {age} years young.") if name == " main ":

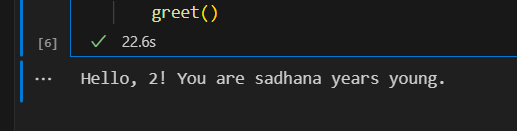
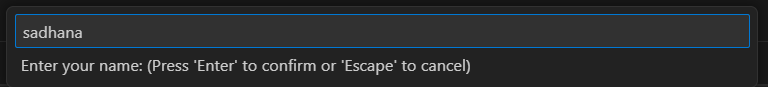
greet()

create a module-level docstring summarizing the purpose, dependencies, and main functions/classes of this Python file write a single multi-line docstring at the top of the file. Ensure the docstring clearly describes functionality and usage without rewriting the entire code.

Code –



Output –



**Task Description #4** (Documentation – Convert Comments to Structured Docstrings)

Use AI to transform existing inline comments into structured function docstrings following Google style.

Prompt –

# Ask the user to enter a number num = int(input("Enter a number: "))

# Initialize result to 1 (since factorial starts from 1) result = 1

# Loop from 1 to the entered number (inclusive)

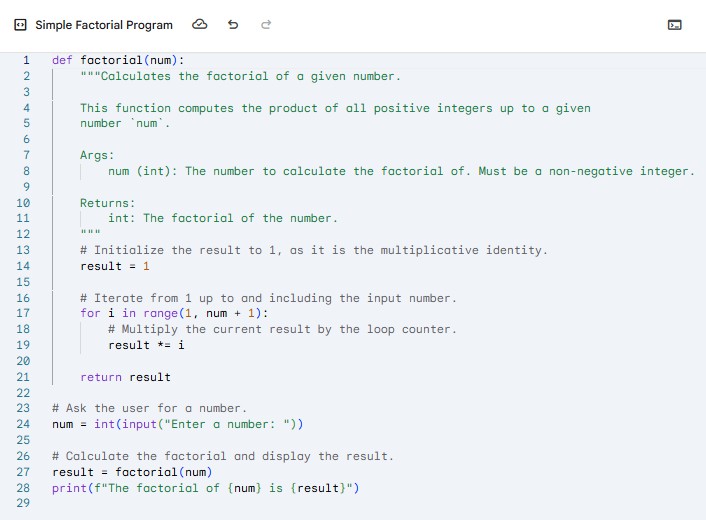
for i in range(1, num + 1):

result \*= i # Multiply result by the current number # Display the final factorial result

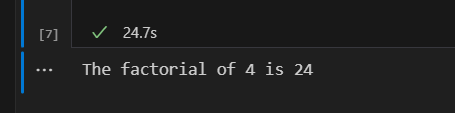
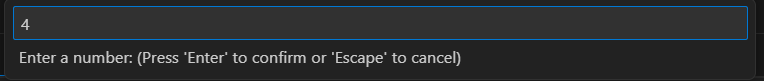
print(f"The factorial of {num} is {result}")

transform existing inline comments into structured function docstrings following Google style move relevant details from comments into function docstrings. Verify that the new docstrings keep the meaning intact while improving structure.

Code –



Output –



**Task Description #5** (Documentation – Review and Correct Docstrings) Use AI to identify and correct inaccuracies in existing docstrings.

Prompt –

def add\_numbers(a, b): """

Multiplies two numbers and returns the result. """

return a + b

def greet\_user(name): """

Calculates the factorial of a number. """

print(f"Hello, {name}!")

def get\_even\_numbers(lst): """

Returns a list of odd numbers from the input list. """

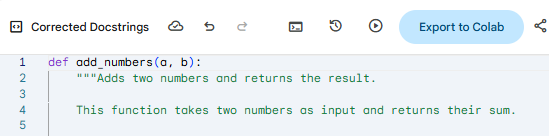
return [x for x in lst if x % 2 == 0] # Example usage

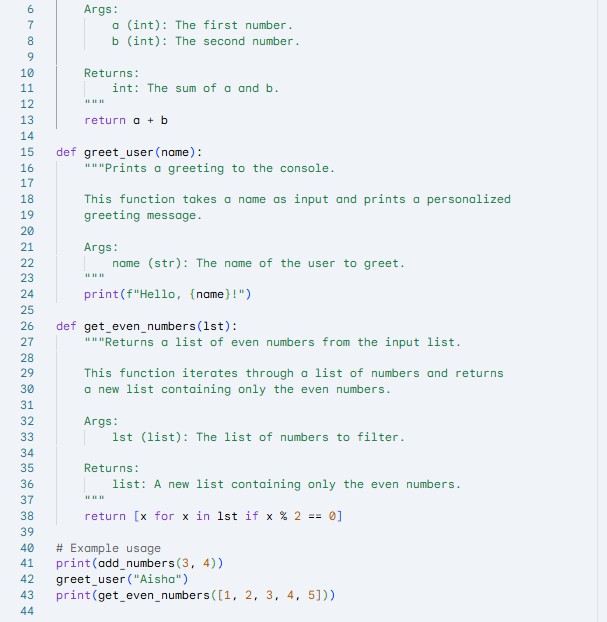
print(add\_numbers(3, 4)) # Actually adds, not multiplies greet\_user("Aisha") # Just prints a greeting

print(get\_even\_numbers([1, 2, 3, 4, 5])) # Returns evens, not odds

dentify and correct inaccuracies in existing docstrings rewrite each docstring to match the current code behavior. Ensure corrections follow Google-style formatting.

Code –





Output –

