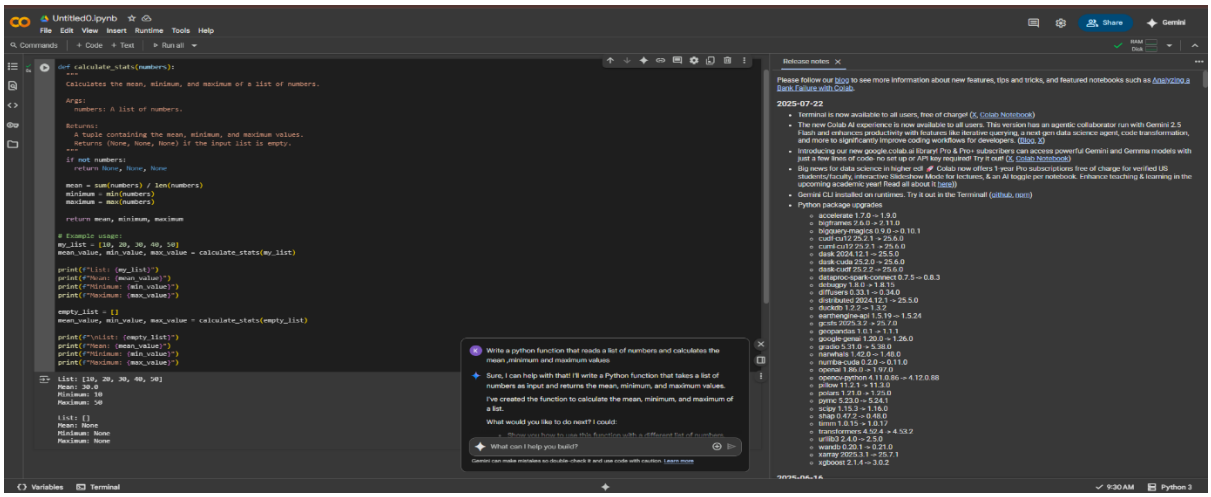
Assignment:2

Name: N. Bhanuchander

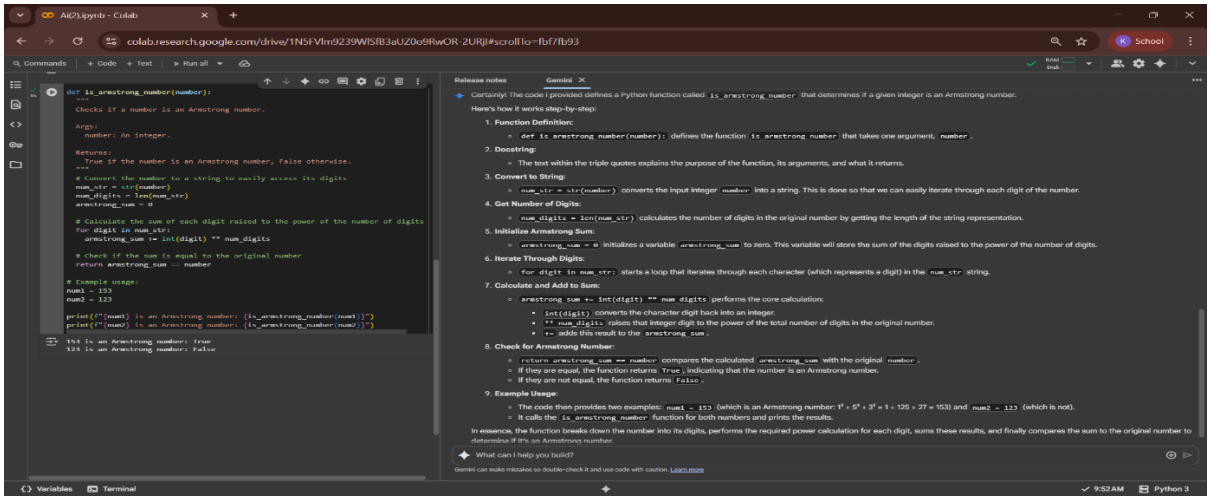
RollNo: 2403a52132

Task:1

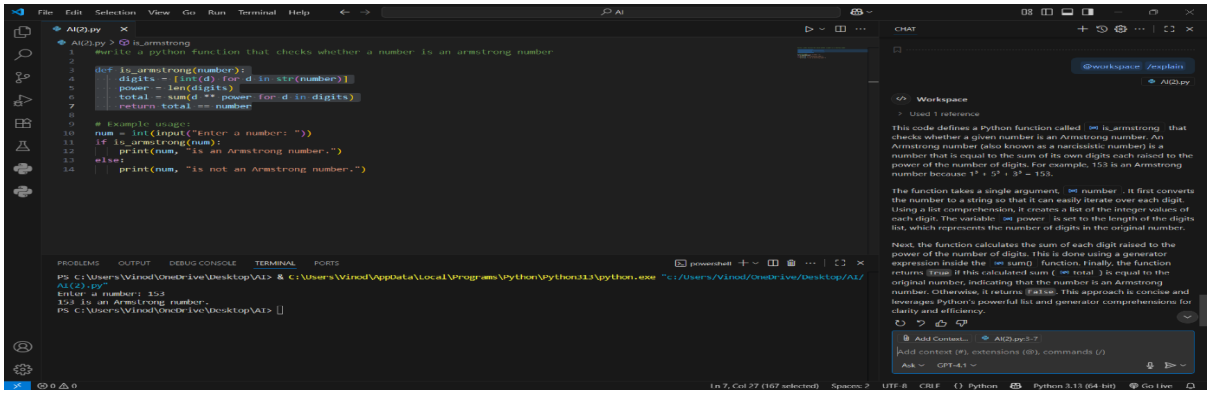


Task:2

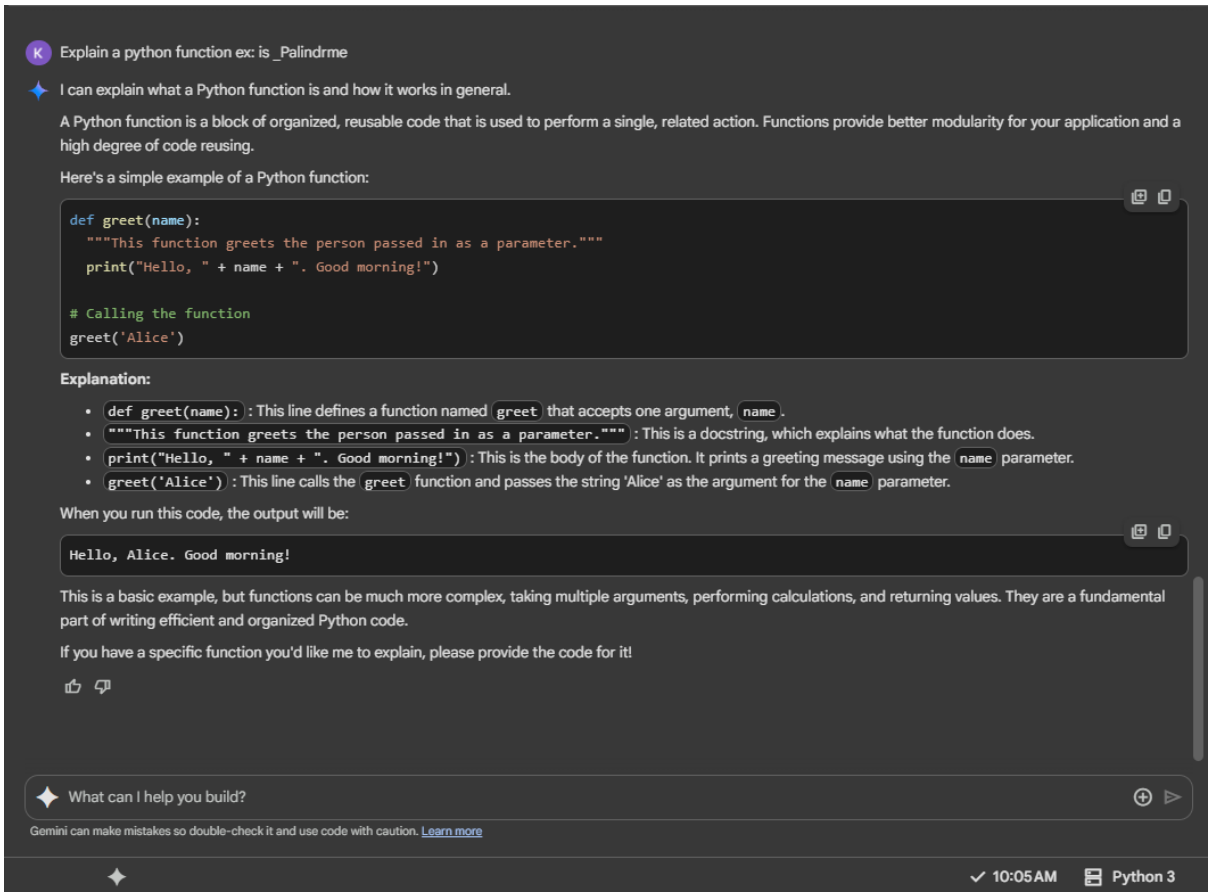
Gemini



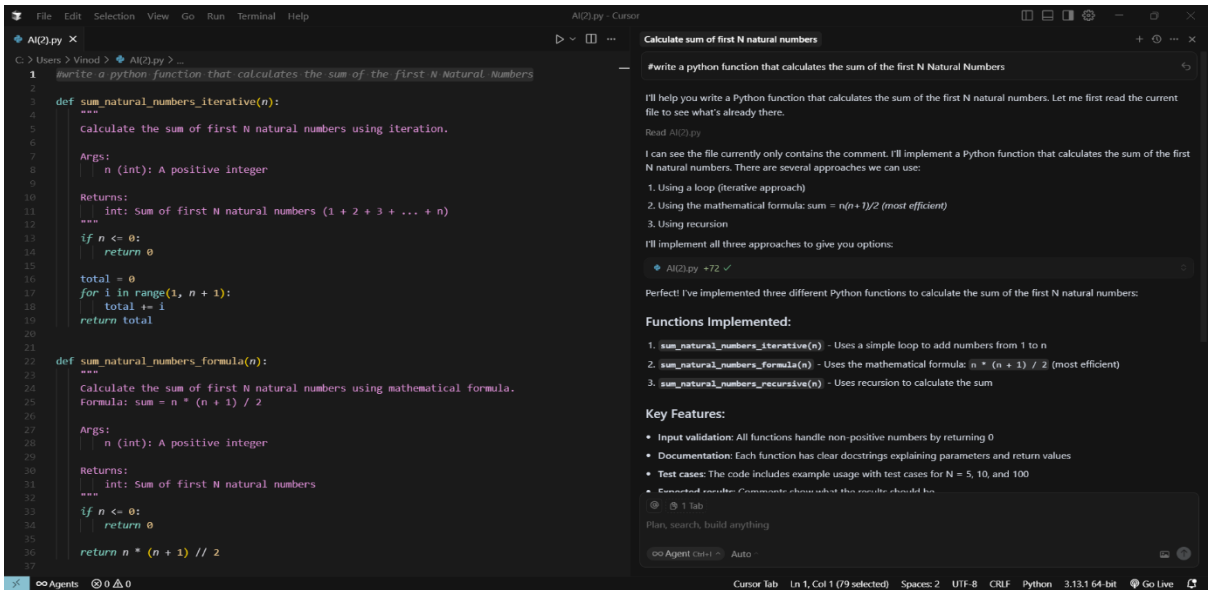
Copilot

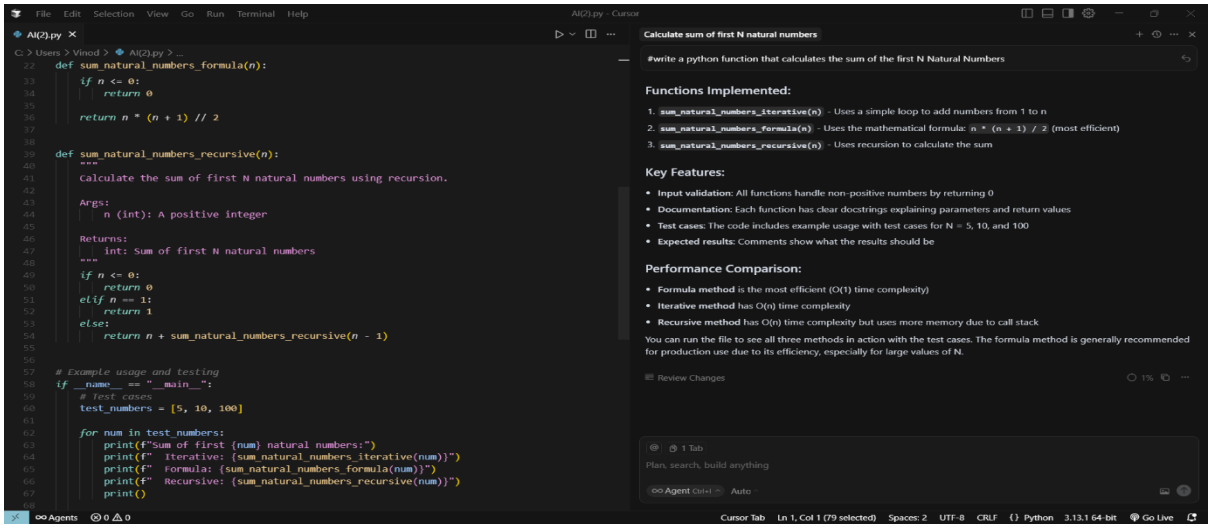


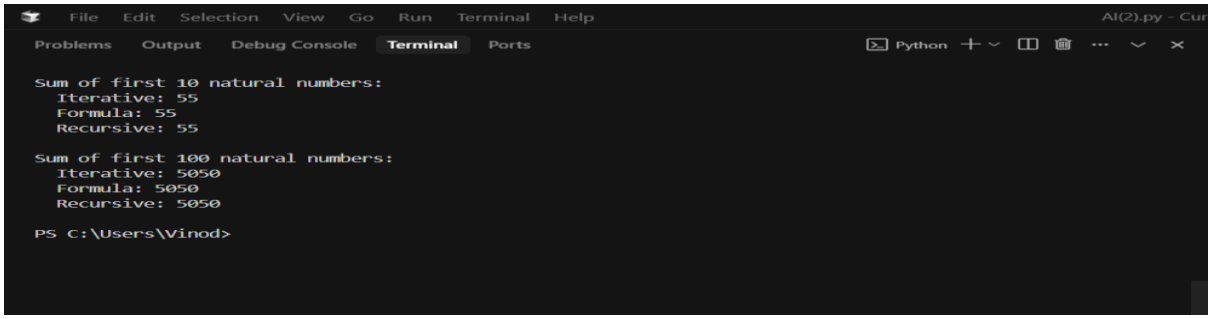
Task:3


Explain a python function ex: is _Palindrme
I can explain what a Python function is and how it works in general.
A Python function is a block of organized, reusable code that is used to perform a single, related action. Functions provide better modularity for your application and a high degree of code reusing.
Here's a simple example of a Python function:
def greet(name):
"""This function greets the person passed in as a parameter."""
print("Hello, " + name + ". Good morning!")
# Calling the function
greet('Alice')
Explanation:
def greet(name): : This line defines a function named greet that accepts one argument, name.
"""This function greets the person passed in as a parameter.""" : This is a docstring, which explains what the function does.
print("Hello, " + name + ". Good morning!") : This is the body of the function. It prints a greeting message using the name parameter.
greet('Alice') : This line calls the greet function and passes the string 'Alice' as the argument for the name parameter.
When you run this code, the output will be:
Hello, Alice. Good morning!
This is a basic example, but functions can be much more complex, taking multiple arguments, performing calculations, and returning values. They are a fundamental part of writing efficient and organized Python code.
If you have a specific function you'd like me to explain, please provide the code for it!
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more
10:05 AM
Python 3

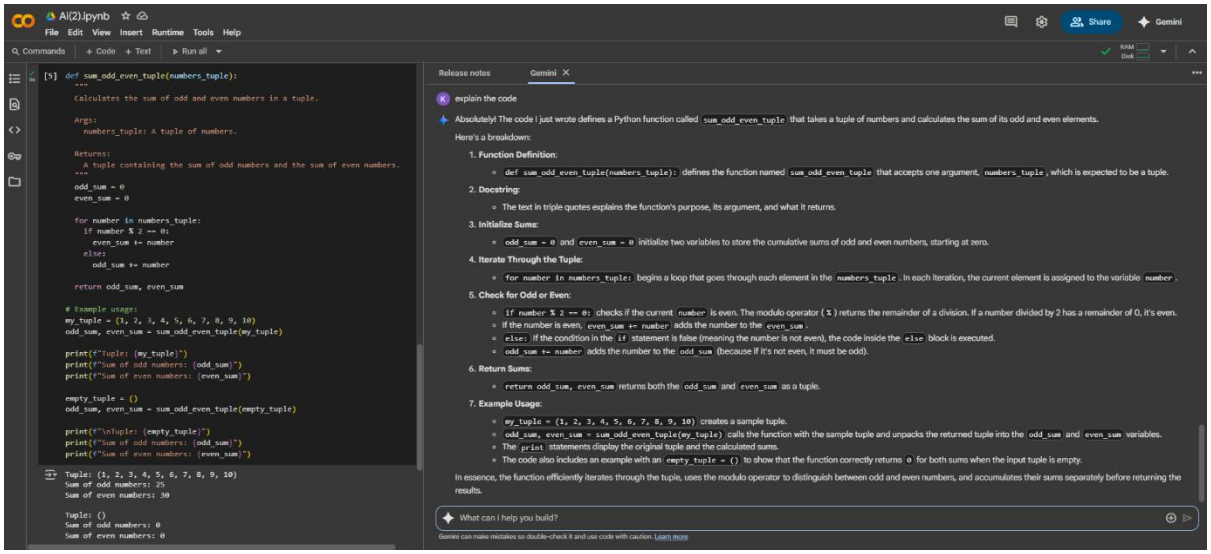
Task:4


File Edit Selection View Go Run Terminal Help
AI(2).py - Cursor
AI(2).py
#write a python function that calculates the sum of the first N Natural Numbers
def sum_natural_numbers_iterative(n):
"""
Calculate the sum of first N natural numbers using iteration.
Args:
n (int): A positive integer
Returns:
int: Sum of first N natural numbers (1 + 2 + 3 + ... + n)
"""
if n <= 0:
return 0
total = 0
for i in range(1, n + 1):
total += i
return total
def sum_natural_numbers_formula(n):
"""
Calculate the sum of first N natural numbers using mathematical formula.
Formula: sum = n * (n + 1) / 2
Args:
n (int): A positive integer
Returns:
int: Sum of first N natural numbers
"""
if n <= 0:
return 0
return n * (n + 1) // 2
Calculate sum of first N natural numbers
#write a python function that calculates the sum of the first N Natural Numbers
I'll help you write a Python function that calculates the sum of the first N natural numbers. Let me first read the current file to see what's already there.
Read AI(2).py
I can see the file currently only contains the comment. I'll implement a Python function that calculates the sum of the first N natural numbers. There are several approaches we can use:
1. Using a loop (iterative approach)
2. Using the mathematical formula: sum = n(n+1)/2 (most efficient)
3. Using recursion
I'll implement all three approaches to give you options:
AI(2).py +72
Perfect! I've implemented three different Python functions to calculate the sum of the first N natural numbers:
Functions Implemented:
1. sum_natural_numbers_iterative(n) - Uses a simple loop to add numbers from 1 to n
2. sum_natural_numbers_formula(n) - Uses the mathematical formula: n * (n + 1) / 2 (most efficient)
3. sum_natural_numbers_recursive(n) - Uses recursion to calculate the sum
Key Features:
Input validation: All functions handle non-positive numbers by returning 0
Documentation: Each function has clear docstrings explaining parameters and return values
Test cases: The code includes example usage with test cases for N = 5, 10, and 100
1 Tab
Plan, search, build anything
Agent Ctrl+I Auto
Agents
Cursor Tab Ln 1, Col 1 (79 selected) Spaces: 2 UTF-8 CRLF Python 3.13.1 64-bit Go Live


AI(2).py
def sum_natural_numbers_formula(n):
if n <= 0:
return 0
return n * (n + 1) // 2
def sum_natural_numbers_recursive(n):
"""
Calculate the sum of first N natural numbers using recursion.
Args:
n (int): A positive integer
Returns:
int: Sum of first N natural numbers
"""
if n <= 0:
return 0
elif n == 1:
return 1
else:
return n + sum_natural_numbers_recursive(n - 1)
# Example usage and testing
if __name__ == "__main__":
# Test cases
test_numbers = [5, 10, 100]
for num in test_numbers:
print(f"Sum of first {num} natural numbers:")
print(f" Iterative: {sum_natural_numbers_iterative(num)}")
print(f" Formula: {sum_natural_numbers_formula(num)}")
print(f" Recursive: {sum_natural_numbers_recursive(num)}")
print()
Calculate sum of first N natural numbers
#write a python function that calculates the sum of the first N Natural Numbers
Functions Implemented:
1. sum_natural_numbers_iterative(n) - Uses a simple loop to add numbers from 1 to n
2. sum_natural_numbers_formula(n) - Uses the mathematical formula: n * (n + 1) / 2 (most efficient)
3. sum_natural_numbers_recursive(n) - Uses recursion to calculate the sum
Key Features:
Input validation: All functions handle non-positive numbers by returning 0
Documentation: Each function has clear docstrings explaining parameters and return values
Test cases: The code includes example usage with test cases for N = 5, 10, and 100
Expected results: Comments show what the results should be
Performance Comparison:
Formula method is the most efficient (O(1) time complexity)
Iterative method has O(n) time complexity
Recursive method has O(n) time complexity but uses more memory due to call stack
You can run the file to see all three methods in action with the test cases. The formula method is generally recommended for production use due to its efficiency, especially for large values of N.
Review Changes
Plan, search, build anything
Agent
Auto


Problems
Output
Debug Console
Terminal
Ports
Python
Sum of first 10 natural numbers:
Iterative: 55
Formula: 55
Recursive: 55
Sum of first 100 natural numbers:
Iterative: 5050
Formula: 5050
Recursive: 5050
PS C:\Users\Vinod>

Task:5


AI(2).ipynb
def sum_odd_even_tuple(numbers_tuple):
"""
Calculates the sum of odd and even numbers in a tuple.
Args:
numbers_tuple: A tuple of numbers.
Returns:
A tuple containing the sum of odd numbers and the sum of even numbers.
"""
odd_sum = 0
even_sum = 0
for number in numbers_tuple:
if number % 2 == 0:
even_sum += number
else:
odd_sum += number
return odd_sum, even_sum
# Example usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
odd_sum, even_sum = sum_odd_even_tuple(my_tuple)
print(f"Tuple: {my_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
empty_tuple = ()
odd_sum, even_sum = sum_odd_even_tuple(empty_tuple)
print(f"\nTuple: {empty_tuple}")
print(f"Sum of odd numbers: {odd_sum}")
print(f"Sum of even numbers: {even_sum}")
Tuple: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
Sum of odd numbers: 25
Sum of even numbers: 30
Tuple: ()
Sum of odd numbers: 0
Sum of even numbers: 0
Release notes
Gemini
explain the code
Absolutely! The code I just wrote defines a Python function called sum_odd_even_tuple that takes a tuple of numbers and calculates the sum of its odd and even elements.
Here's a breakdown:
1. Function Definition:
def sum_odd_even_tuple(numbers_tuple): defines the function named sum_odd_even_tuple that accepts one argument, numbers_tuple, which is expected to be a tuple.
2. Docstring:
The text in triple quotes explains the function's purpose, its argument, and what it returns.
3. Initialize Sums:
odd_sum = 0 and even_sum = 0 initialize two variables to store the cumulative sums of odd and even numbers, starting at zero.
4. Iterate Through the Tuple:
for number in numbers_tuple: begins a loop that goes through each element in the numbers_tuple. In each iteration, the current element is assigned to the variable number.
5. Check for Odd or Even:
if number % 2 == 0: checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's even.
If the number is even, even_sum += number adds the number to the even_sum.
else: If the condition in the if statement is false (meaning the number is not even), the code inside the else block is executed.
odd_sum += number adds the number to the odd_sum (because if it's not even, it must be odd).
6. Return Sums:
return odd_sum, even_sum returns both the odd_sum and even_sum as a tuple.
7. Example Usage:
my_tuple = (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) creates a sample tuple.
odd_sum, even_sum = sum_odd_even_tuple(my_tuple) calls the function with the sample tuple and unpacks the returned tuple into the odd_sum and even_sum variables.
The print statements display the original tuple and the calculated sums.
The code also includes an example with an empty_tuple = () to show that the function correctly returns 0 for both sums when the input tuple is empty.
In essence, the function efficiently iterates through the tuple, uses the modulo operator to distinguish between odd and even numbers, and accumulates their sums separately before returning the results.
What can I help you build?
Gemini can make mistakes so double-check it and use code with caution. Learn more