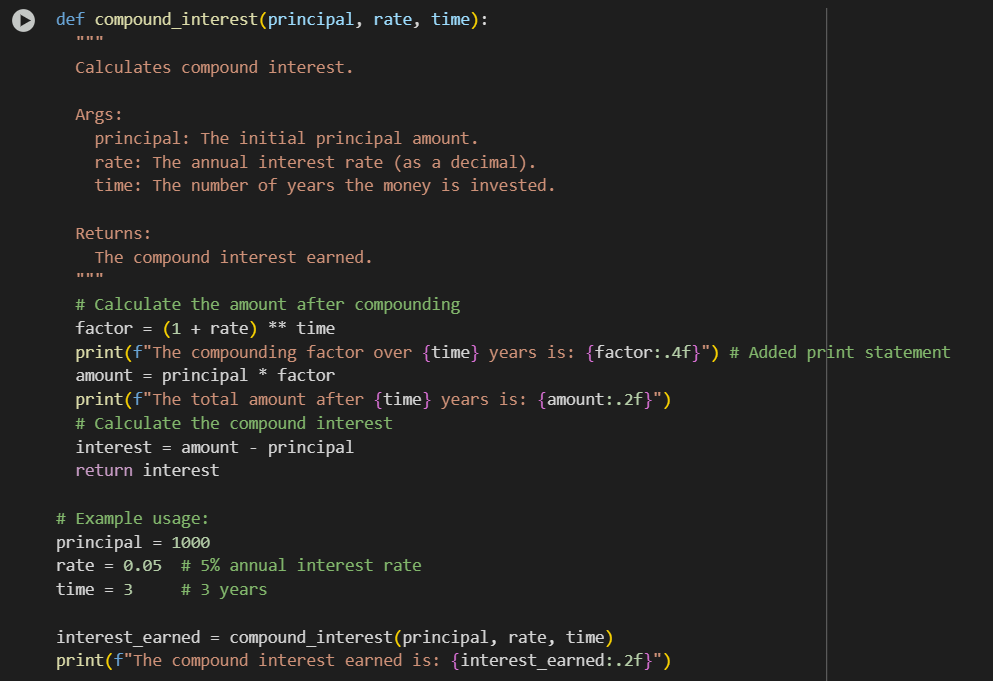
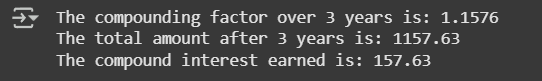
**Task Description#1**

Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example

Code:



Output:



Explanation:

def compound\_interest(principal, rate, time):: This line defines a function named compound\_interest that accepts three arguments: principal (the initial investment), rate (the annual interest rate as a decimal), and time (the number of years).

""" Calculates compound interest. ... Returns: The compound interest earned. """: This is a docstring, which explains what the function does, its arguments, and what it returns.

factor = (1 + rate) \*\* time: This line calculates the compounding factor. It adds 1 to the rate (representing the original principal plus the interest) and raises it to the power of time.

print(f"The compounding factor over {time} years is: {factor:.4f}"): This line prints the calculated compounding factor, formatted to four decimal places. It shows how much the initial principal will grow by after the specified time due to compounding.

amount = principal \* factor: This line calculates the total amount of money after the interest is compounded by multiplying the initial principal by the calculated factor.

print(f"The total amount after {time} years is: {amount:.2f}"): This line prints the total calculated amount after the specified number of years, formatted to two decimal places.

interest = amount - principal: This line calculates the compound interest earned by subtracting the original principal from the final amount.

return interest: This line sends the calculated interest value back as the result of the function.

principal = 1000: This line sets the value of the principal variable to 1000 for the example.

rate = 0.05: This line sets the value of the rate variable to 0.05 (representing 5%) for the example.

time = 3: This line sets the value of the time variable to 3 (representing 3 years) for the example.

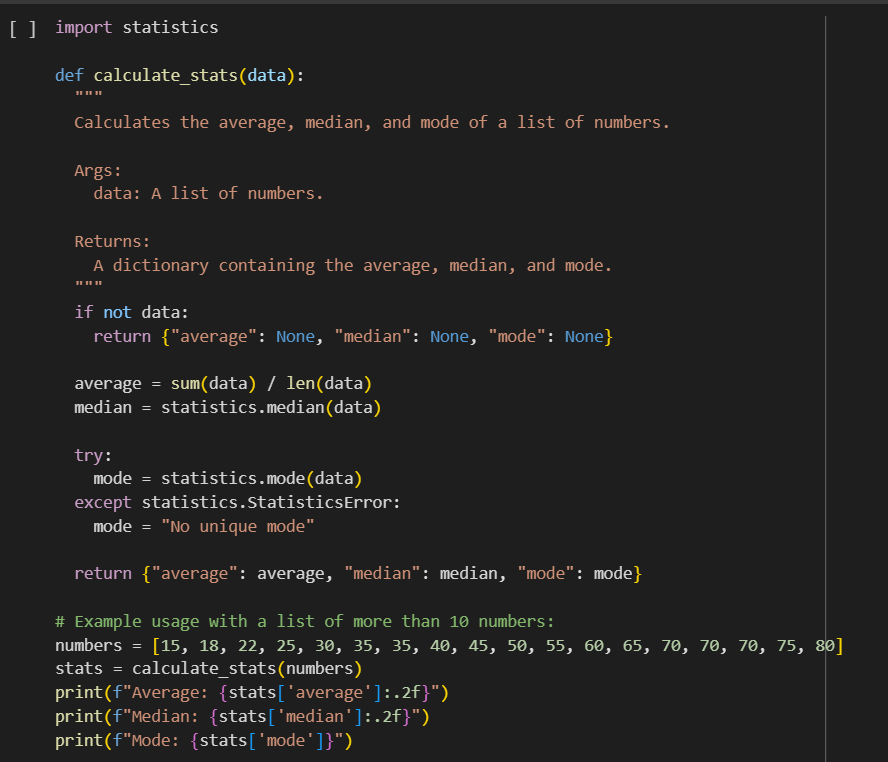
interest\_earned = compound\_interest(principal, rate, time): This line calls the compound\_interest function with the example values and stores the returned interest in the interest\_earned variable.

print(f"The compound interest earned is: {interest\_earned:.2f}"): This line prints the final calculated compound interest earned, formatted to two decimal places

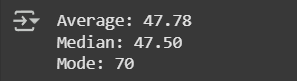
**Task Description#2**

Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers

Code:



Output:



Explanation:

import statistics: This line imports the statistics module, which provides functions for calculating mathematical statistics of numeric data, such as the median and mode.

def calculate\_stats(data):: This line defines a function named calculate\_stats that takes one argument: data. This data is expected to be a list of numbers for which we want to calculate statistics.

""" Calculates the average, median, and mode of a list of numbers. ... Returns: A dictionary containing the average, median, and mode. """: This is a multiline string called a docstring. It explains what the function does, describes its input argument (Args), and specifies what the function returns (Returns).

if not data:: This line checks if the input list data is empty. An empty list is considered "falsy" in Python, so not data will be True if the list is empty.

return {"average": None, "median": None, "mode": None}: If the if condition is True (meaning the list is empty), this line is executed. It returns a dictionary with the keys "average", "median", and "mode", and sets their values to None. This indicates that these statistics cannot be calculated for an empty list.

average = sum(data) / len(data): This line calculates the average. sum(data) calculates the sum of all numbers in the list, and len(data) gets the number of elements in the list. The sum is then divided by the count to get the average.

median = statistics.median(data): This line calculates the median of the list data using the median() function from the imported statistics module. The median is the middle value of a sorted list.

try:: This keyword starts a try block. Code within a try block is executed, and if an error occurs, the code in the corresponding except block is executed. This is used here to handle potential errors when calculating the mode.

mode = statistics.mode(data): This line attempts to calculate the mode (the most frequently occurring value) of the list data using the mode() function from the statistics module.

except statistics.StatisticsError:: This line specifies that if a StatisticsError occurs within the try block, the code within this except block should be executed. A StatisticsError can happen if the list has no unique mode (e.g., if all elements appear the same number of times, or if there are multiple modes).

mode = "No unique mode": If a StatisticsError is caught, this line sets the mode variable to the string "No unique mode".

return {"average": average, "median": median, "mode": mode}: This line returns a dictionary containing the calculated average, median, and mode.

# Example usage with a list of more than 10 numbers:: This is a comment indicating the start of the example usage section of the code.

numbers = [15, 18, 22, 25, 30, 35, 35, 40, 45, 50, 55, 60, 65, 70, 70, 70, 75, 80]: This line creates a list named numbers containing 18 integers (more than 10). This list is used as an example input for the calculate\_stats function.

stats = calculate\_stats(numbers): This line calls the calculate\_stats function with the numbers list as input. The dictionary returned by the function (containing the calculated average, median, and mode) is stored in the variable stats.

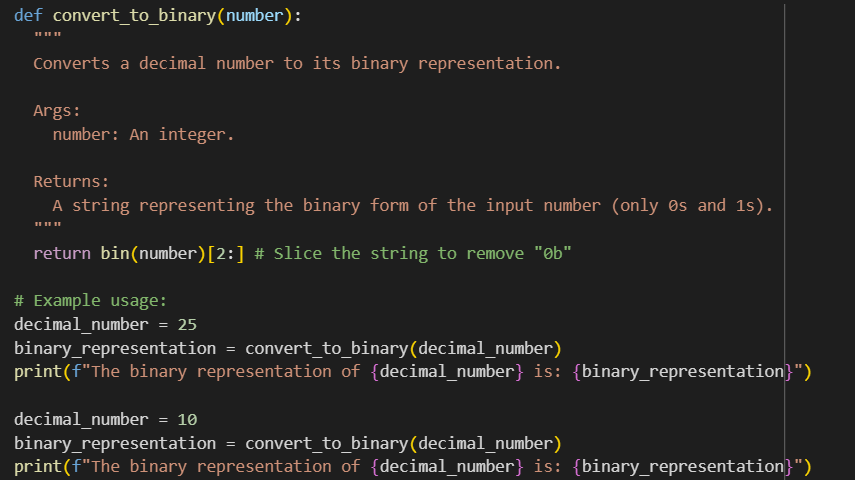
print(f"Average: {stats['average']:.2f}"): This line prints the calculated average. It uses an f-string to include the label "Average: " and the value of the average retrieved from the stats dictionary (stats['average']). The :.2f is a format specifier that formats the number to two decimal places.

print(f"Median: {stats['median']:.2f}"): This line prints the calculated median, similarly formatted to two decimal places using an f-string.

print(f"Mode: {stats['mode']}"): This line prints the calculated mode. It retrieves the mode from the stats dictionary and prints it using an f-string. Note that the mode is not formatted to two decimal places because it might be a string ("No unique mode")

**Task Description#3**

* Provide multiple examples of input-output to the AI for convert\_to\_binary(num) function. Observe how AI uses few-shot prompting to generalize.



Output:



Explanation:  
def convert\_to\_binary(number):: This line defines a function named convert\_to\_binary that takes one argument: number, which is expected to be an integer (the decimal number you want to convert).

""" Converts a decimal number to its binary representation. ... (only 0s and 1s). """: This is a docstring. It explains what the function does, describes its input argument (Args), and specifies what the function returns (Returns). It also clarifies that the output will only contain 0s and 1s.

return bin(number)[2:]: This is the core of the function and where the conversion and formatting happen:

bin(number): This part calls the built-in Python function bin(). It takes the input number and converts it into a string that represents its binary form. By default, bin() prefixes this string with "0b". For example, bin(25) returns "0b11001".

[2:]: This is string slicing in Python. It takes the string returned by bin(number) and creates a new string that starts from the character at index 2 and goes to the end of the original string. This effectively removes the first two characters ("0b"). So, "0b11001"[2:] results in "11001".

return: This keyword sends the resulting sliced string (the binary representation without "0b") back as the output of the convert\_to\_binary function.

# Example usage:: This is a comment indicating the start of the example usage section of the code.

decimal\_number = 25: This line sets the value of the variable decimal\_number to 25 for the first example.

binary\_representation = convert\_to\_binary(decimal\_number): This line calls the convert\_to\_binary function with decimal\_number (which is 25). The function returns the binary string "11001", which is then stored in the variable binary\_representation.

print(f"The binary representation of {decimal\_number} is: {binary\_representation}"): This line prints the output for the first example. It uses an f-string to include the original decimal\_number (25) and its binary\_representation ("11001") in a readable sentence.

decimal\_number = 10: This line sets the value of the variable decimal\_number to 10 for the second example.

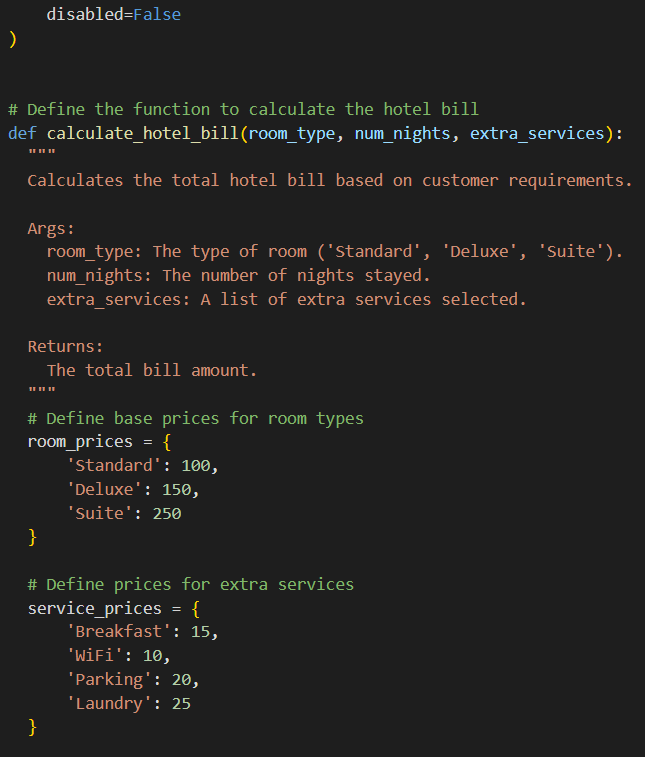
binary\_representation = convert\_to\_binary(decimal\_number): This line calls the convert\_to\_binary function with decimal\_number (which is 10). The function returns the binary string "1010", which is then stored in the variable binary\_representation.

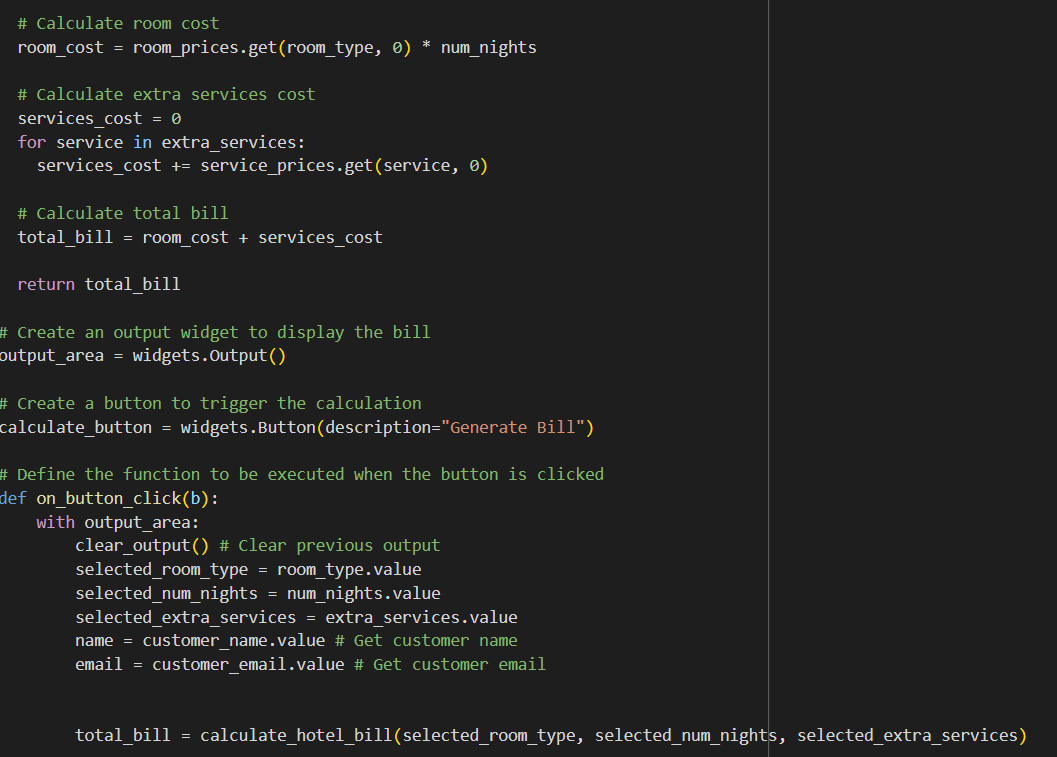
print(f"The binary representation of {decimal\_number} is: {binary\_representation}"): This line prints the output for the second example, displaying the original decimal\_number (10) and its binary\_representation ("1010").

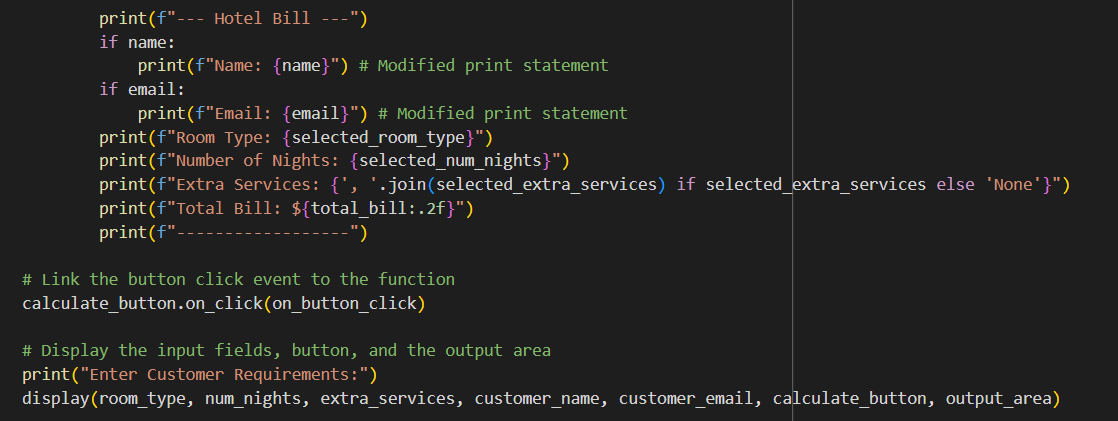
**Task Description#4**

* Create an user interface for an hotel to generate bill based on customer requirements









Output:

