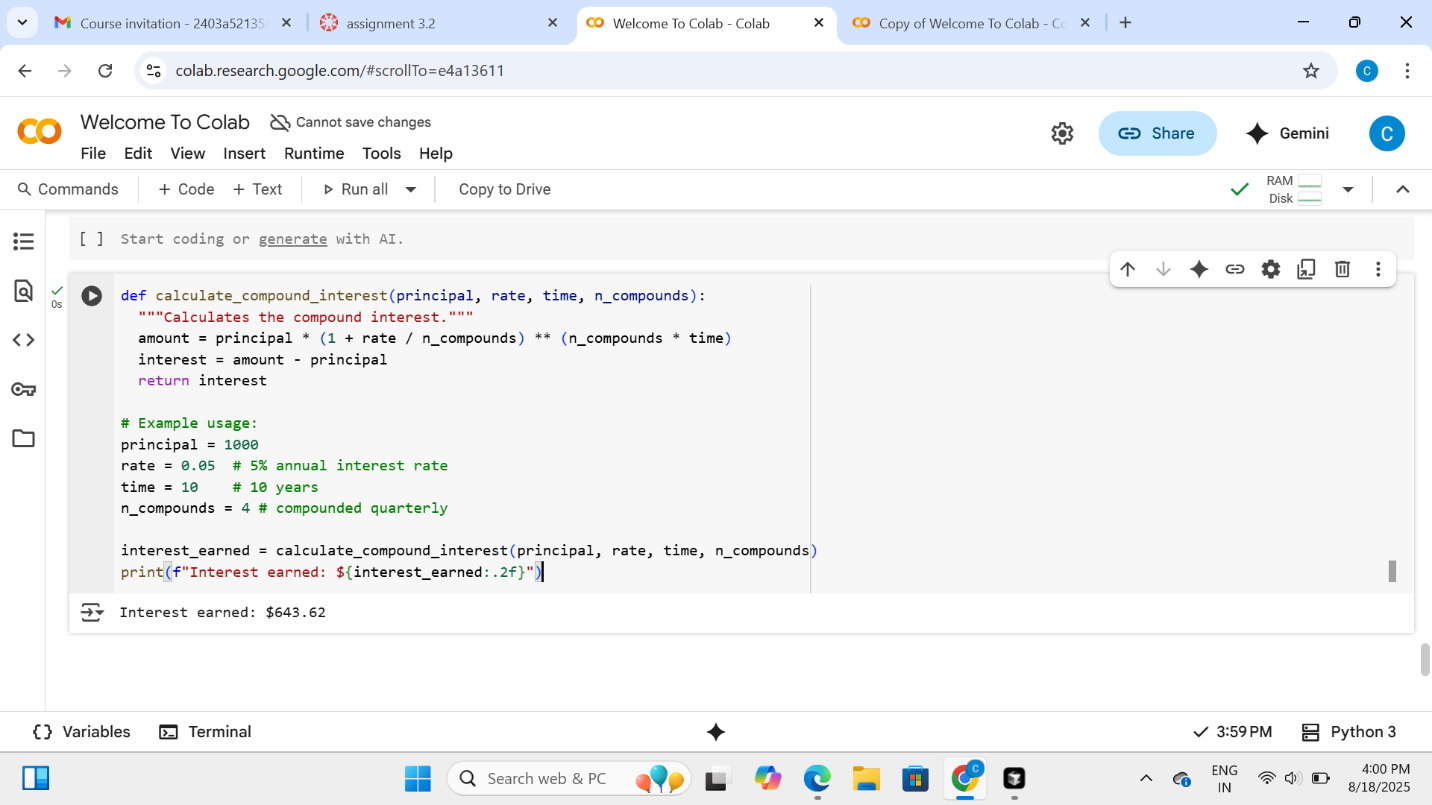
Assignment-3

Task-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 Expected Output#1

● Comparison of AI-generated code styles



Explanation:

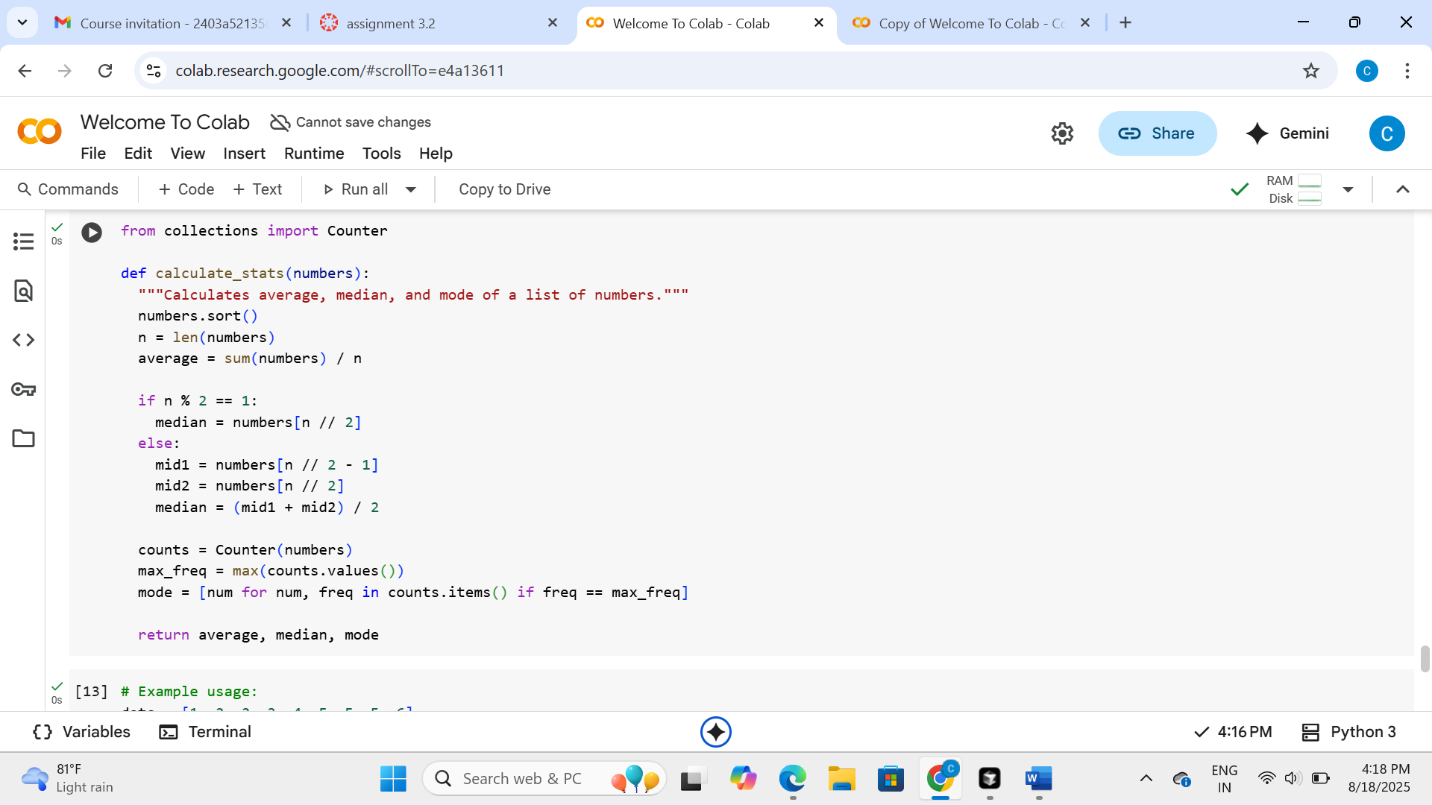
* **def calculate\_compound\_interest(principal, rate, time, n\_compounds):**: This line defines a function named calculate\_compound\_interest that takes four arguments: principal, rate, time, and n\_compounds.
* **"""Calculates the compound interest."""**: This is the docstring, which explains what the function does.
* **amount = principal \* (1 + rate / n\_compounds) \*\* (n\_compounds \* time)**: This line calculates the total amount of money after the compound interest is applied, using the standard compound interest formula.
* **interest = amount - principal**: This line calculates the interest earned by subtracting the initial principal from the total amount.
* **return interest**: This line returns the calculated interest earned.

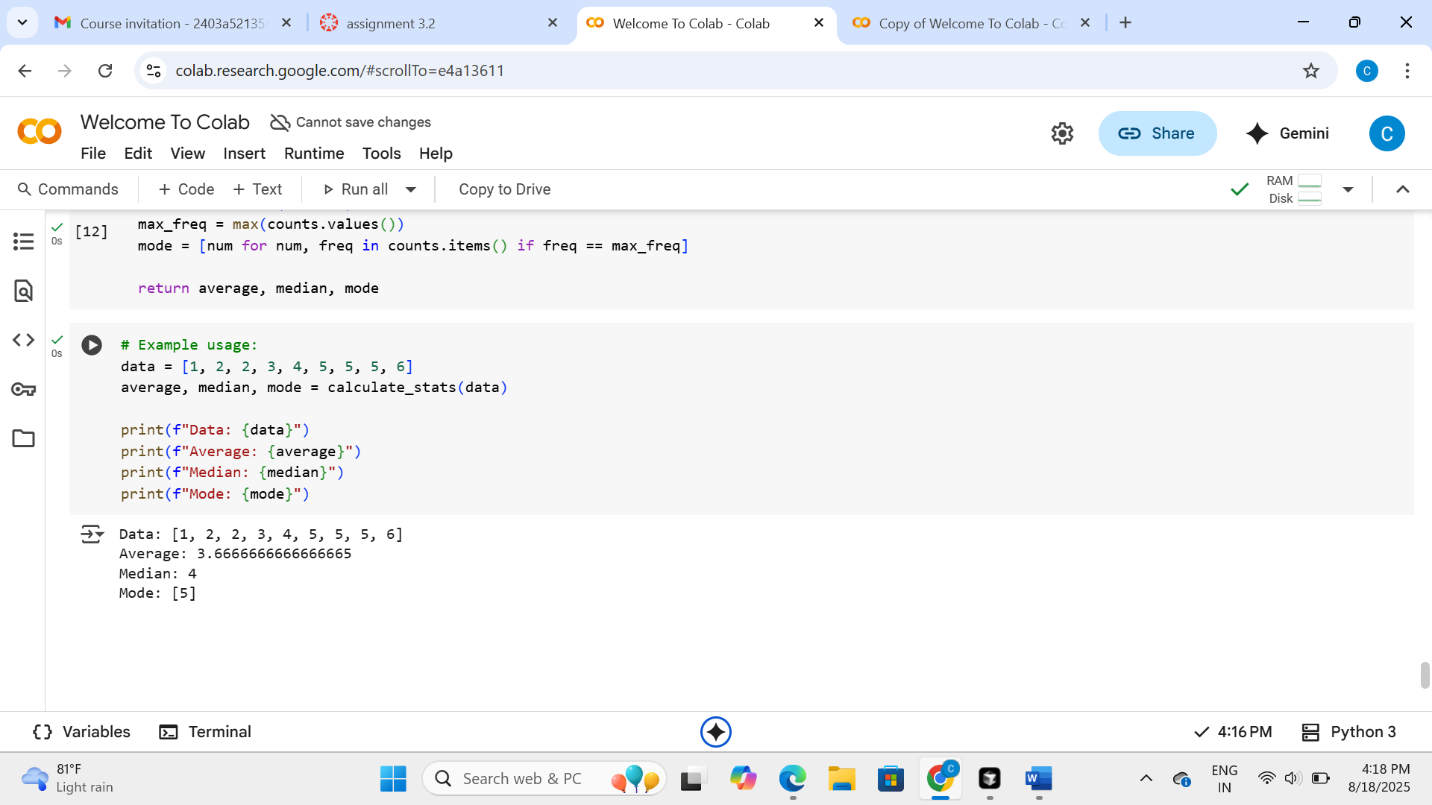
The example usage demonstrates how to call the function with sample values and prints the calculated interest.

Task-2

Do math stuff, then refine it to:

# Write a function to calculate average, median, and  
mode of a list of numbers.  
Expected Output#2  
● AI-generated function evolves from unclear to accurate multi-statistical operation





Explanation:

1. **Average:** This is like sharing everything equally. You add up all the numbers and then divide by how many numbers there are.
2. **Median:** If you line up all the numbers from smallest to biggest, the median is the number right in the middle. If there are two numbers in the middle, you take the average of those two.
3. **Mode:** This is the number that appears most often in the list. There might be one mode, or there might be a few numbers that appear the same most often.

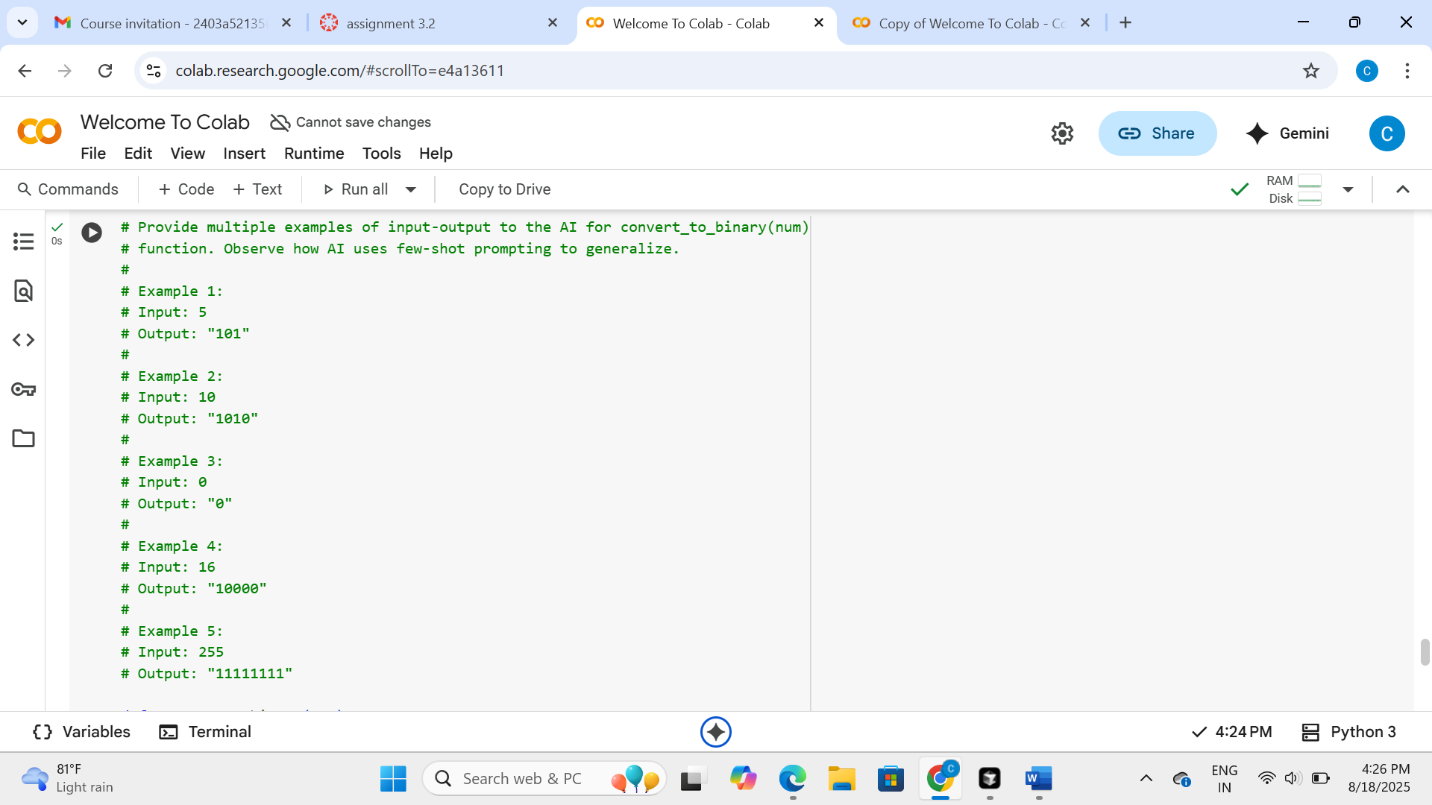
The code does these three things:

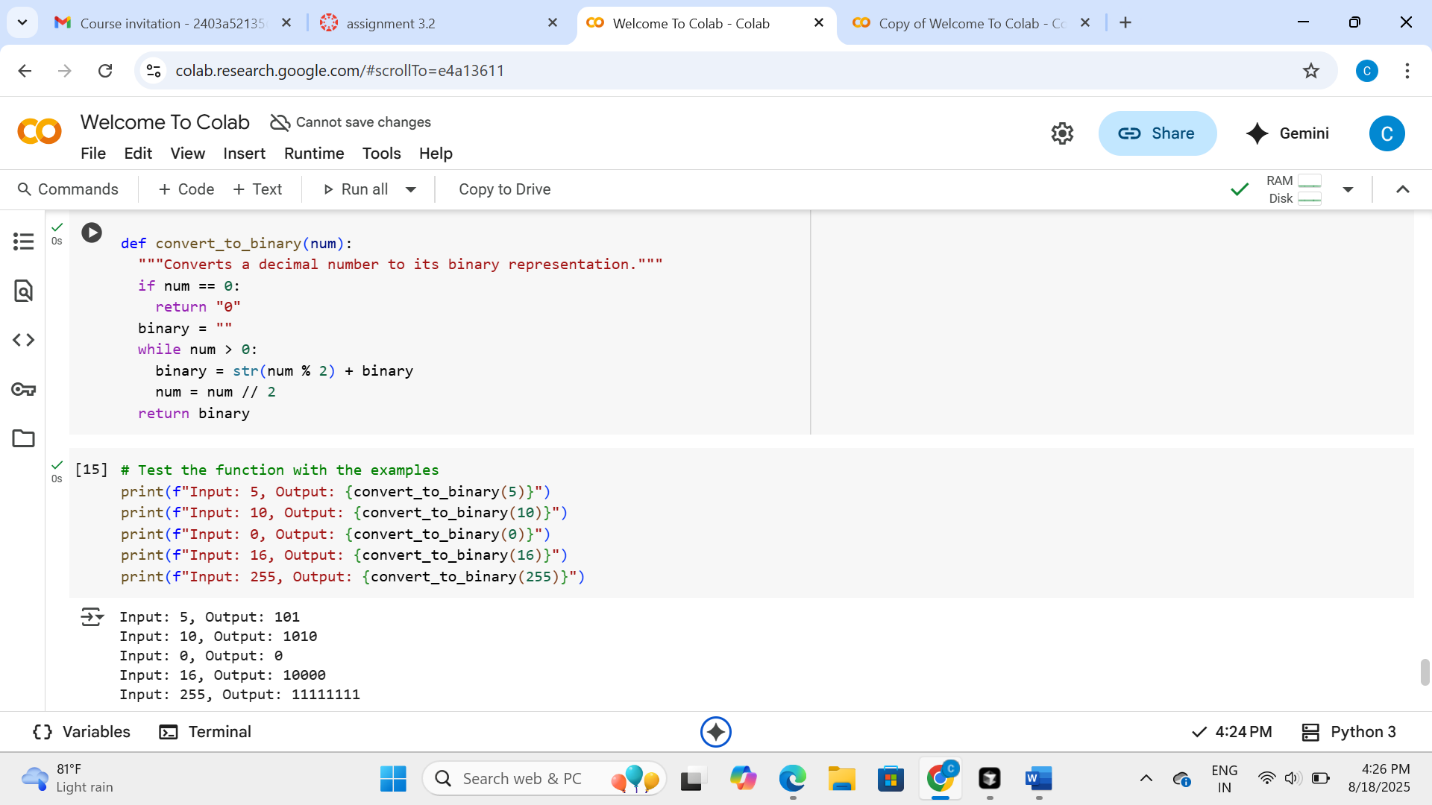
* It first sorts the list of numbers so they are in order.
* Then it calculates the average by adding them all up and dividing.
* It finds the middle number(s) to figure out the median.
* Finally, it counts how many times each number shows up and finds the one(s) that show up the most, which is the mode.

The function then gives you back these three results: the average, the median, and the mode.

Task-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.  
Expected Output#3  
● Enhanced AI output with clearer prompts





Explanation:

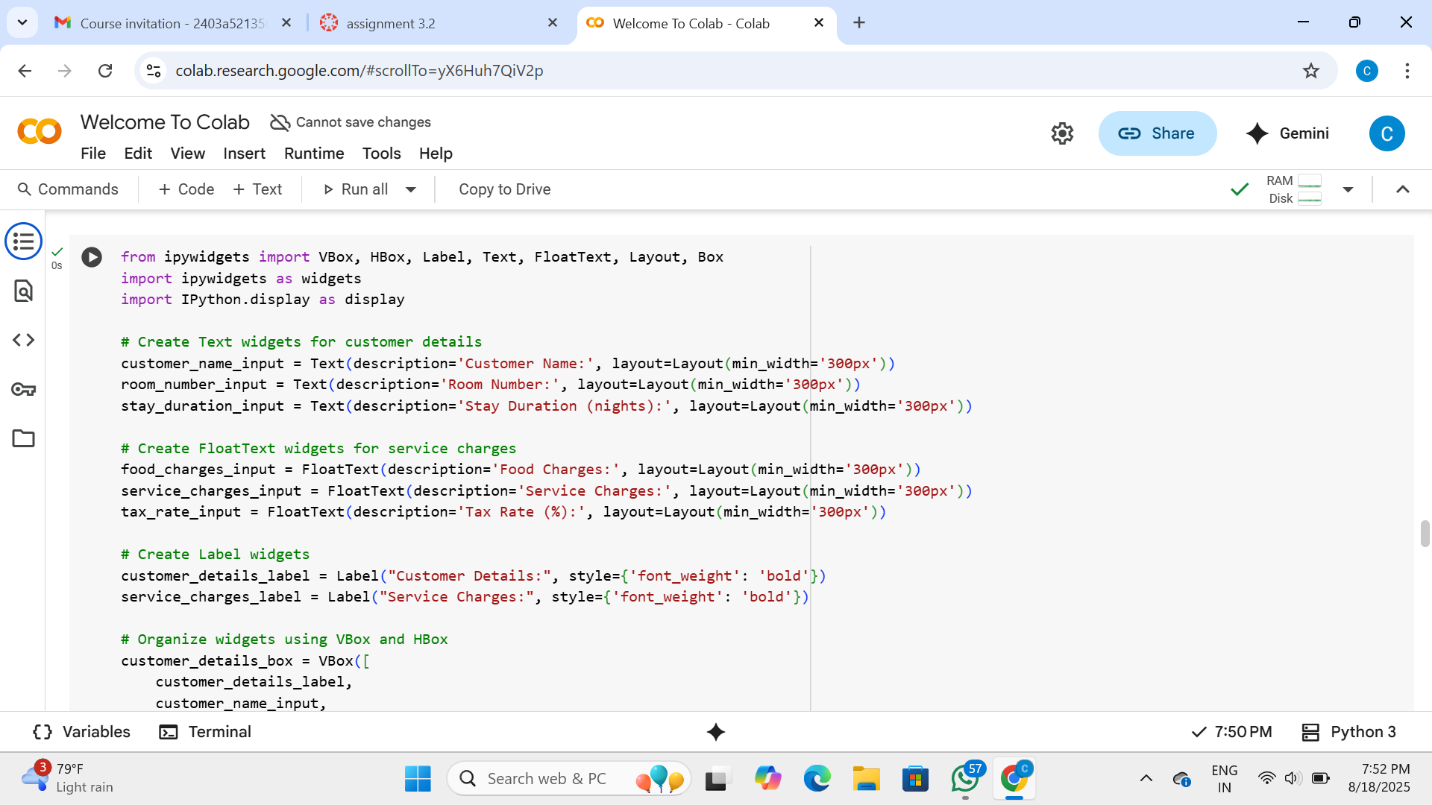
Convert\_to\_binary function is like a translator that takes a regular number (the kind we use every day) and turns it into a "binary" number. Binary numbers only use two digits: 0 and 1. Computers use binary numbers to understand things.

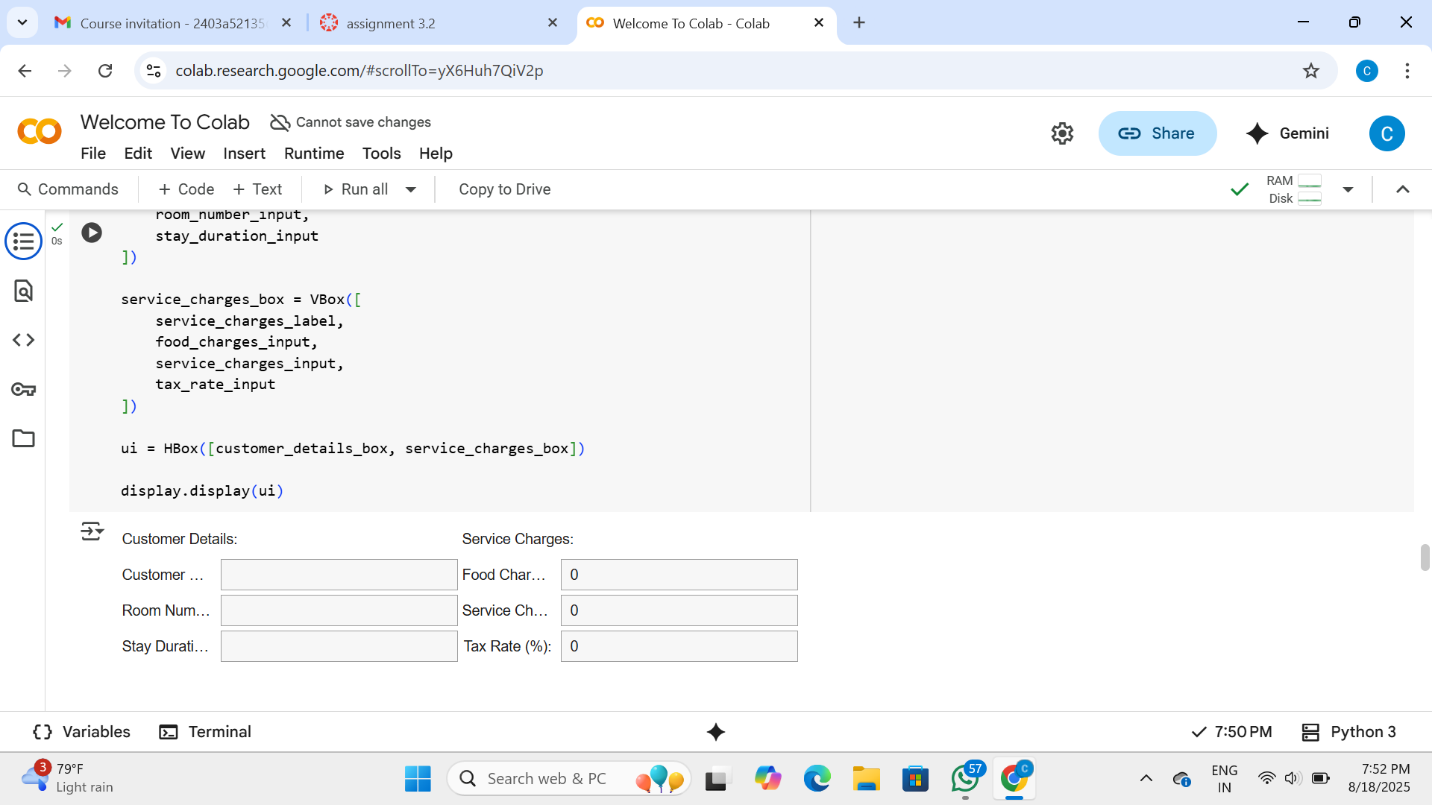
Here's how the code does it, step-by-step:

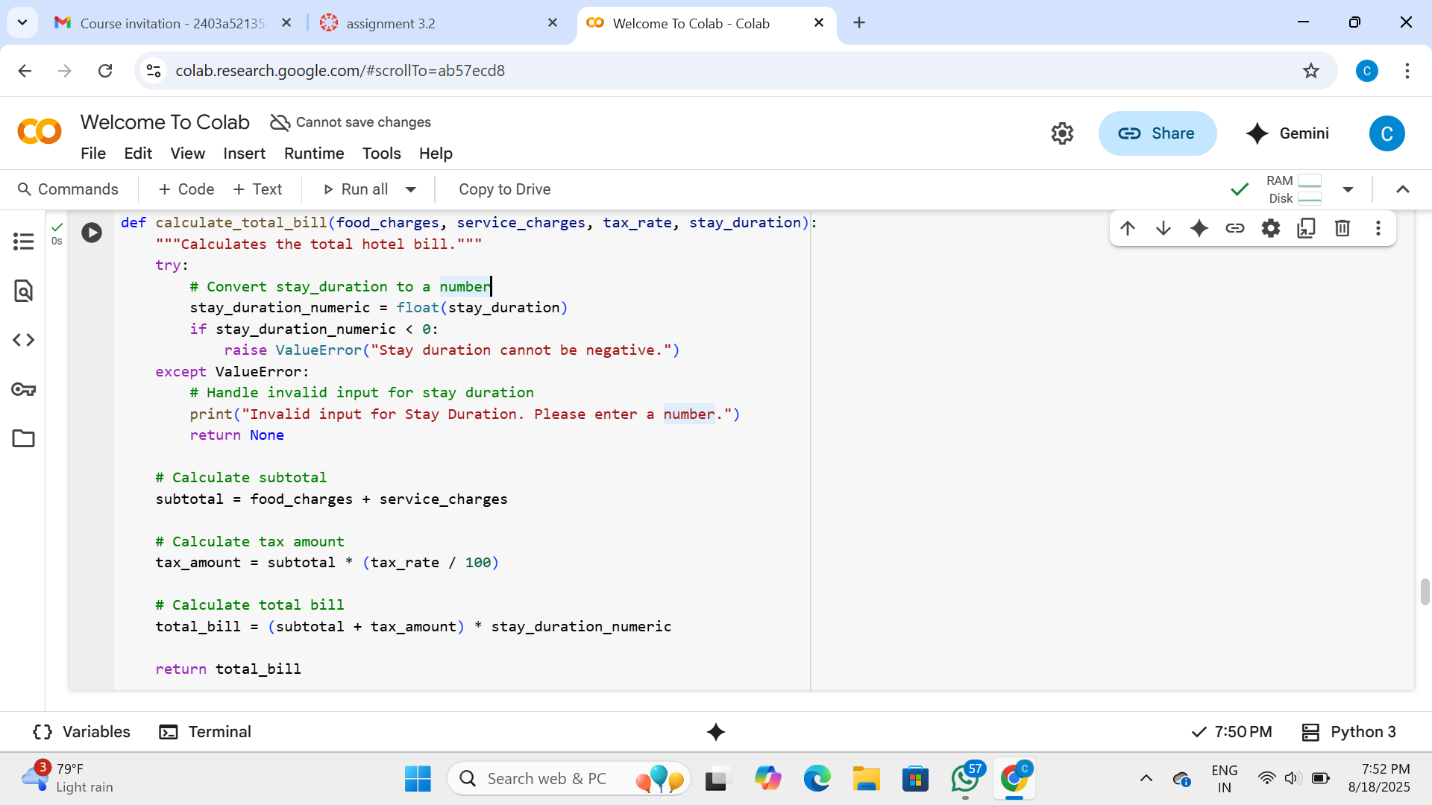
1. **It checks if the number is zero:** If the number you give it is just 0, it knows the binary is also just "0" and stops there.
2. **It gets ready to build the binary number:** It starts with an empty box (a variable called binary) where it will put the binary digits.
3. **It works while there's still something left:** It keeps doing a calculation as long as the original number is bigger than zero.
4. **It finds the last binary digit:** It does a little math trick (num % 2) to see if the current number is even or odd. If it's odd, the binary digit is 1; if it's even, the binary digit is 0. It adds this digit to the *front* of the binary box.
5. **It gets ready for the next digit:** It then divides the original number by 2 (and throws away any remainder) to get ready to find the next binary digit.
6. **It repeats until the number is zero:** It keeps doing steps 4 and 5 until the original number becomes 0.
7. **It gives you the binary number:** Once the number is 0, the binary box has the complete binary version of your original number, and the function gives it back to you.

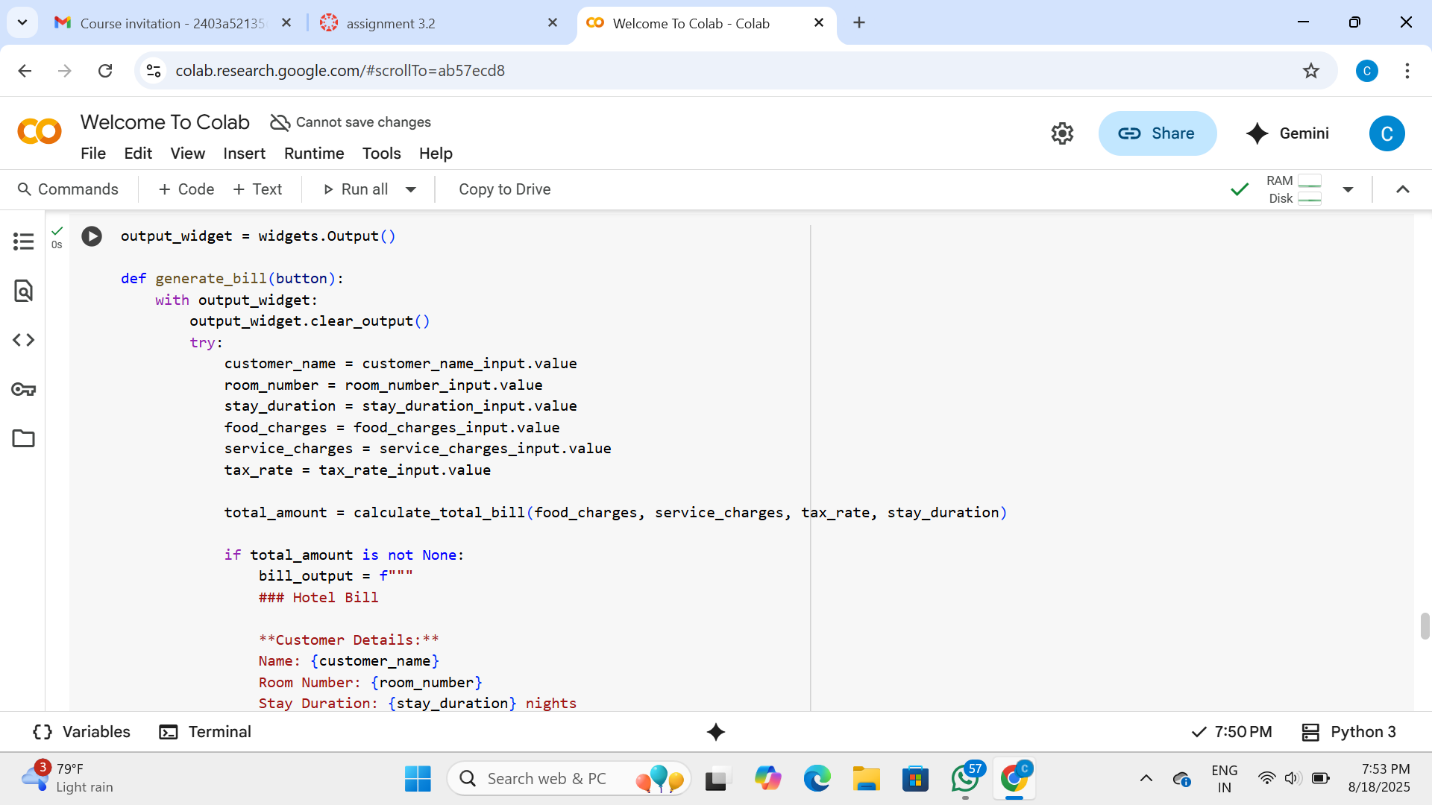
So, in short, it's like repeatedly asking "is this number even or odd?" and using the answer (0 or 1) to build the binary number from right to left.

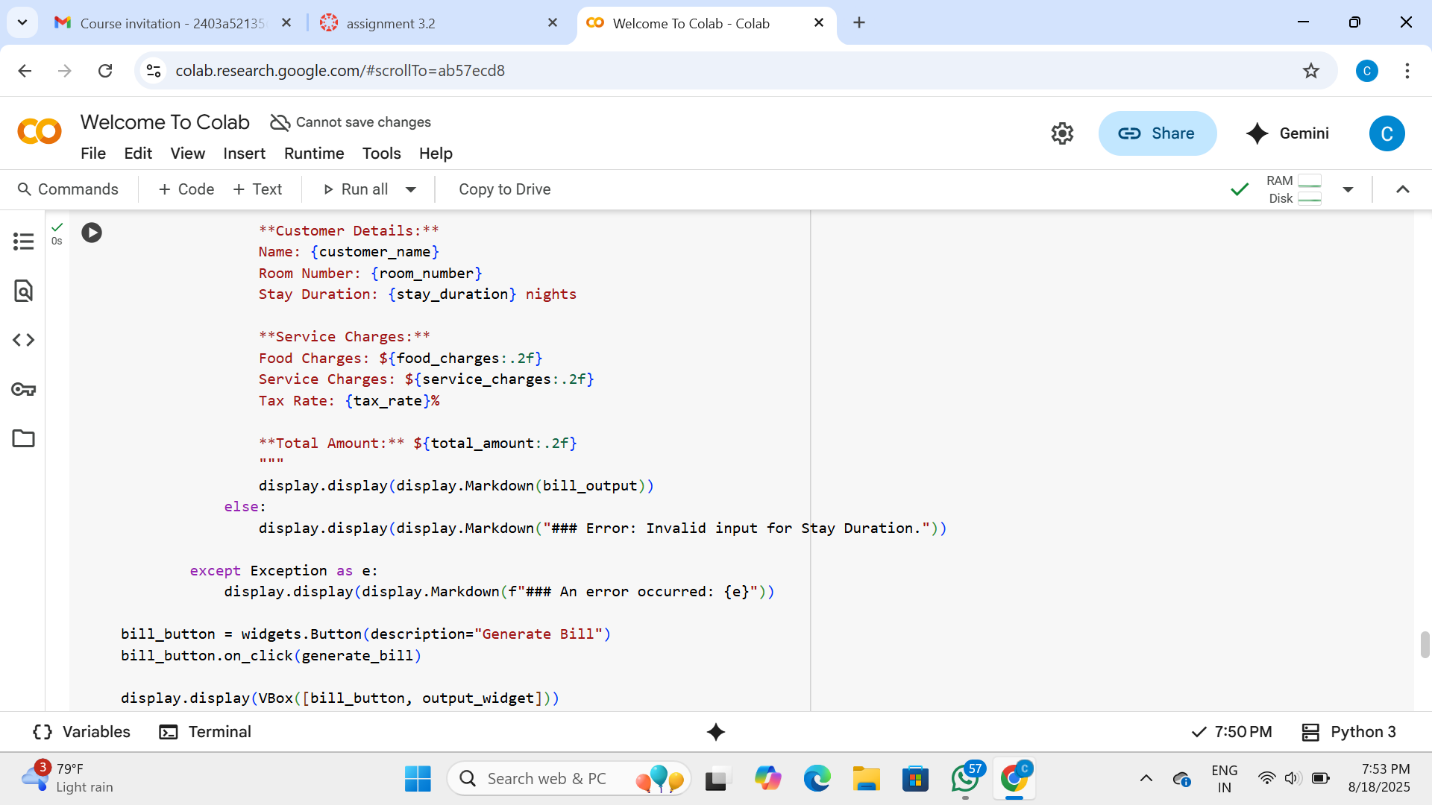
Task-4











Explanation:

magine you have a few different things you need to calculate for a bill, like the cost of the room per night, the cost of extra services, and maybe even taxes.

* **Functions:** Think of functions as little machines that do a specific job. We can create a "room cost machine" (a function), a "service cost machine" (another function), and a "tax machine" (another function). You give them some information (like how many nights or the cost of a service), and they give you back the answer.
* **Consistent functions:** This means that all our "machines" (functions) for calculating things should work in a similar way. They should expect the same kind of information and give back the answer in a similar format. This makes it easier to understand how to use them and makes the code less confusing.
* **Shared logic:** This means if we have parts of the calculation that are the same for different things, we only write that code once and have our different "machines" (functions) use it. For example, if both calculating the room cost and the service cost involve multiplying a price by a quantity, we could have a little helper piece of code that both of them use instead of writing that multiplication code twice.

So, "Consistent functions with shared logic" just means we'll organize our code into clear, easy-to-use "machines" (functions) that work similarly and share any common steps to avoid repeating ourselves. This makes the code neater, easier to fix if something goes wrong, and simpler to add new features later.

Task-5

