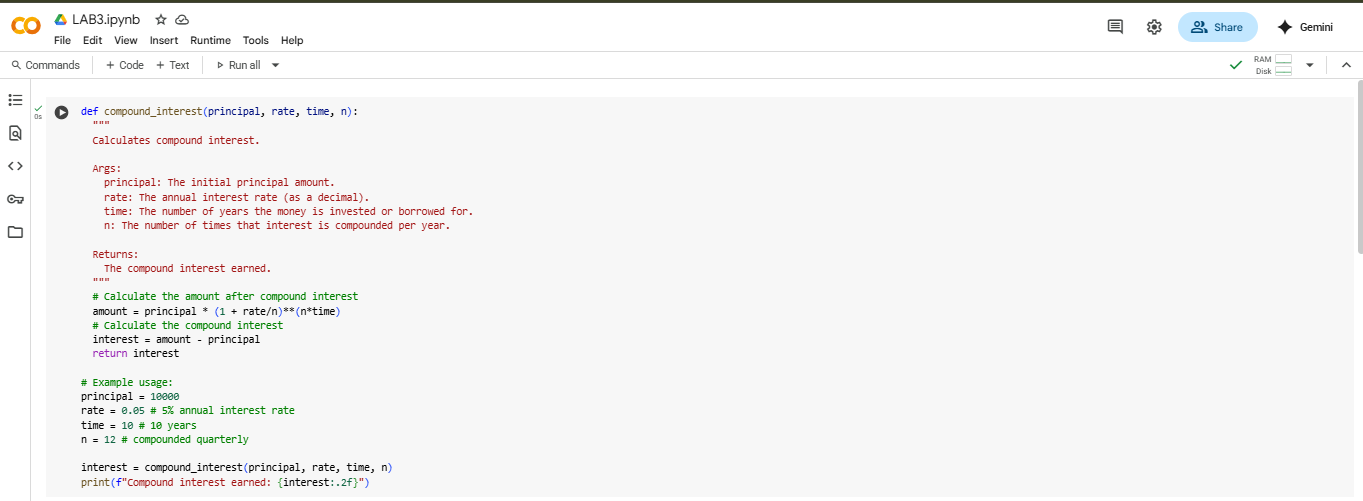
**ASSIGNMENT – 3**

**Task – 1:**

write a python program to find compound interest using functions

**Code:**



**Output:**



**Explanation:**

Here's a breakdown:

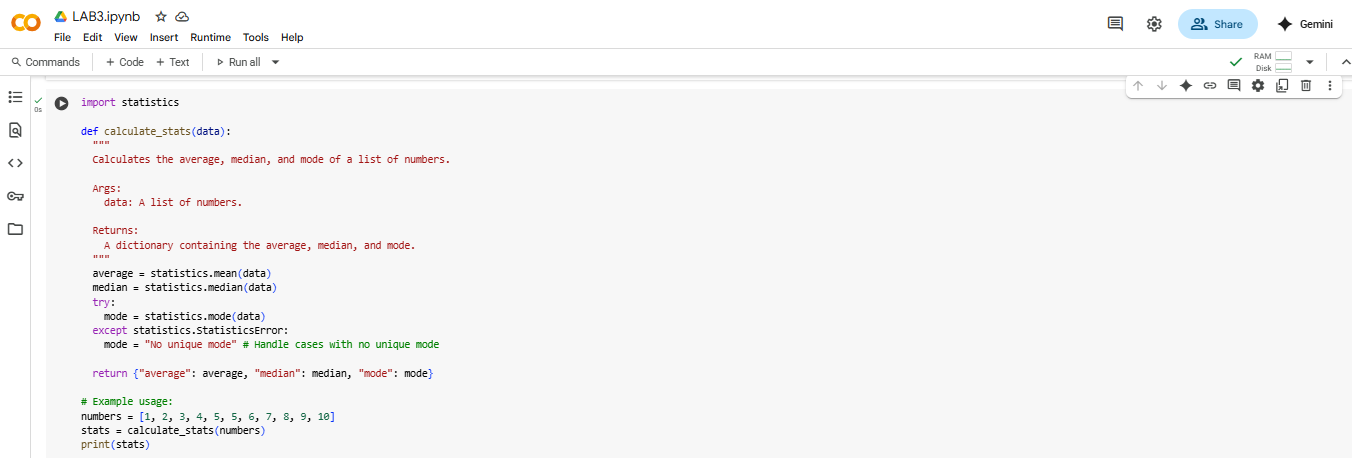
* **def compound\_interest(principal, rate, time, n):**: This line defines the function compound\_interest and specifies that it takes four arguments: principal, rate, time, and n.
* **`amount = principal \* (1 + rate/n)**(n*time)`\**: This line calculates the total amount of the investment after the specified time, using the compound interest formula.
  + principal: The initial amount invested.
  + rate: The annual interest rate (as a decimal).
  + time: The number of years the money is invested.
  + n: The number of times the interest is compounded per year.
* **interest = amount - principal**: This line calculates the compound interest earned by subtracting the initial principal from the total amount.
* **return interest**: This line returns the calculated compound interest.

The code then provides an example of how to use the function with a principal of 10000, a rate of 5%, a time of 10 years, and interest compounded monthly (n=12). Finally, it prints the calculated compound interest, formatted to two decimal places.

**Task – 2:**

write a python function to calculate average median and mode of a list of numbers

**Code:**



**Output:**



**Explanation:**

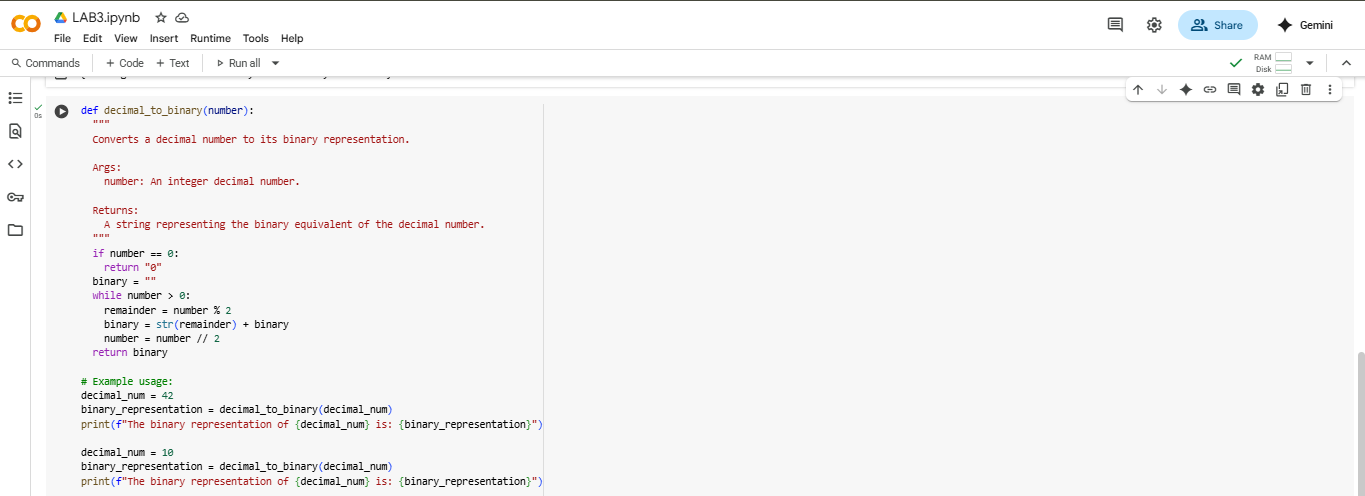
The code in cell 2377c95c defines a Python function called analyze\_numbers that calculates the average, median, and mode of a list of numbers. Here's a breakdown of how it works:

1. from collections import Counter: This line imports the Counter class from the collections module. Counter is a specialized dictionary subclass for counting hashable objects. It's used here to easily count the occurrences of each number in the input list to find the mode.
2. def analyze\_numbers(numbers):: This defines the function analyze\_numbers that takes one argument, numbers, which is expected to be a list of numbers.
3. if not numbers:: This checks if the input list numbers is empty. If it is, it returns a dictionary with None for average, median, and mode, as these calculations are not possible on an empty list.
4. average = sum(numbers) / len(numbers): This calculates the average (mean) of the numbers by summing all the numbers in the list and dividing by the total count of numbers in the list.
5. sorted\_numbers = sorted(numbers): This creates a new list called sorted\_numbers by sorting the input numbers list in ascending order. Sorting is necessary to calculate the median.
6. n = len(sorted\_numbers): This gets the number of elements in the sorted list and stores it in the variable n.
7. if n % 2 == 0:: This checks if the number of elements n is even. o If it's even, the median is the average of the two middle elements. sorted\_numbers[n // 2 - 1] gets the element before the middle, and sorted\_numbers[n // 2] gets the element at the middle. Their sum is divided by 2 to get the median. o else:: If the number of elements n is odd, the median is the middle element. sorted\_numbers[n // 2] gets the middle element using integer division.
8. counts = Counter(numbers): This uses the Counter class to create a dictionary-like object where keys are the numbers from the input list and values are their frequencies (how many times they appear).
9. max\_count = max(counts.values()): This finds the highest frequency (the count of the most frequent number(s)) from the counts dictionary.
10. mode = [num for num, count in counts.items() if count == max\_count]: This creates a list called mode containing all numbers that have a frequency equal to max\_count. This handles cases with multiple modes.
11. if len(mode) == len(numbers):: This checks if every number in the input list appears the same number of times (i.e., every number is a mode). In this case, there is no unique mode, so mode is set to the string "No unique mode".
12. elif len(mode) == 1:: This checks if there is only one number with the highest frequency. If so, that single number is the unique mode, and mode is set to that number.
13. return {"average": average, "median": median, "mode": mode}: Finally, the function returns a dictionary containing the calculated average, median, and mode. The code then provides three examples of how to use the analyze\_numbers function with different lists and prints the results.

**Task – 3:**

write a python program to convert a number into binary by using functions

**Code:**



**Output:**



**Explanation:**

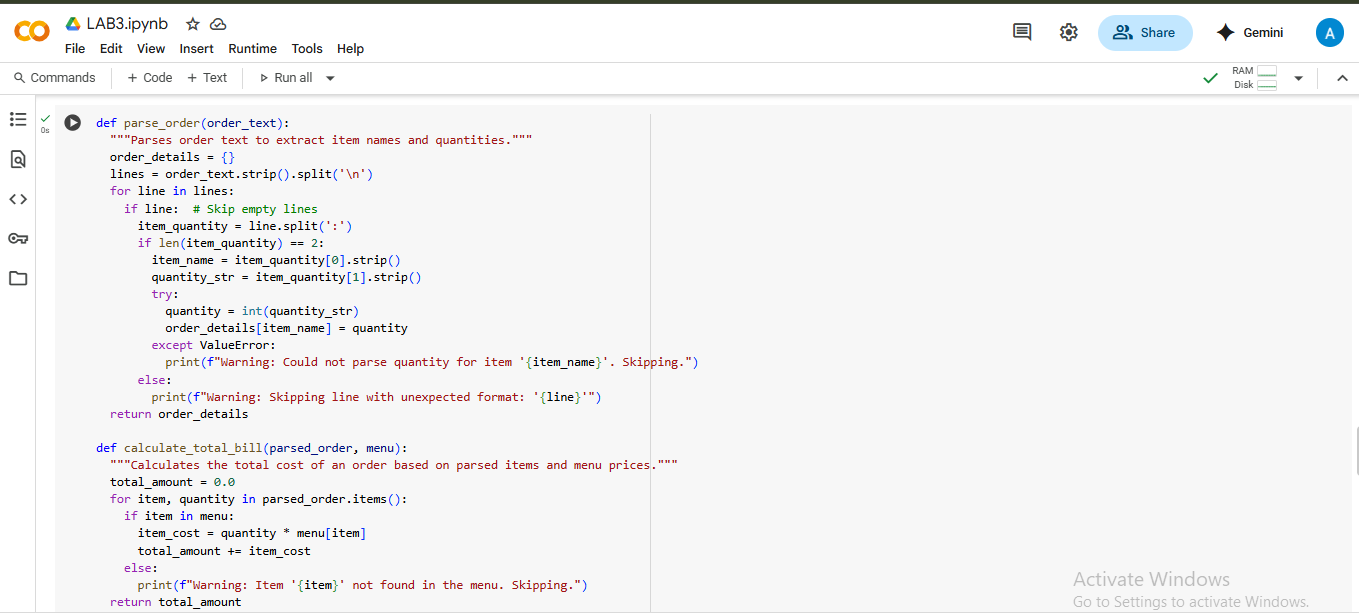
The code in cell bda67346 defines a Python function called decimal\_to\_binary that converts a non-negative integer from decimal (base 10) to binary (base 2). Here's a step-by-step explanation:

1. def decimal\_to\_binary(number):: This line defines the function decimal\_to\_binary that takes one argument, number, which is the decimal number you want to convert.
2. if not isinstance(number, int) or number < 0:: This is a check to ensure the input is valid. It verifies if the number is not an integer or if it's negative. If either condition is true, it means the input is not a non-negative integer that this function is designed to handle, so it returns the string "Invalid input".
3. if number == 0:: This handles the special case where the input number is 0. The binary representation of 0 is simply "0", so it returns "0".
4. binary\_representation = "": An empty string is initialized to store the binary digits as they are calculated.
5. temp\_number = number: A temporary variable temp\_number is created and assigned the value of the input number. This is done so that the original input number is not modified during the conversion process.
6. while temp\_number > 0:: This is the main loop that performs the conversion. It continues as long as temp\_number is greater than 0.
7. remainder = temp\_number % 2: Inside the loop, the modulo operator (%) is used to get the remainder when temp\_number is divided by 2. The remainder will always be either 0 or 1, which are the digits in binary. This remainder is the least significant bit of the remaining decimal number.
8. binary\_representation = str(remainder) + binary\_representation: The remainder (converted to a string) is prepended (added to the beginning) to the binary\_representation string. This builds the binary number from right to left (least significant digit to most significant digit).
9. temp\_number = temp\_number // 2: The temp\_number is then updated by performing integer division (//) by 2. This effectively shifts the decimal number to the right by one binary place, preparing for the calculation of the next binary digit.
10. return binary\_representation: Once the while loop finishes (when temp\_number becomes 0), the function returns the complete binary\_representation string.

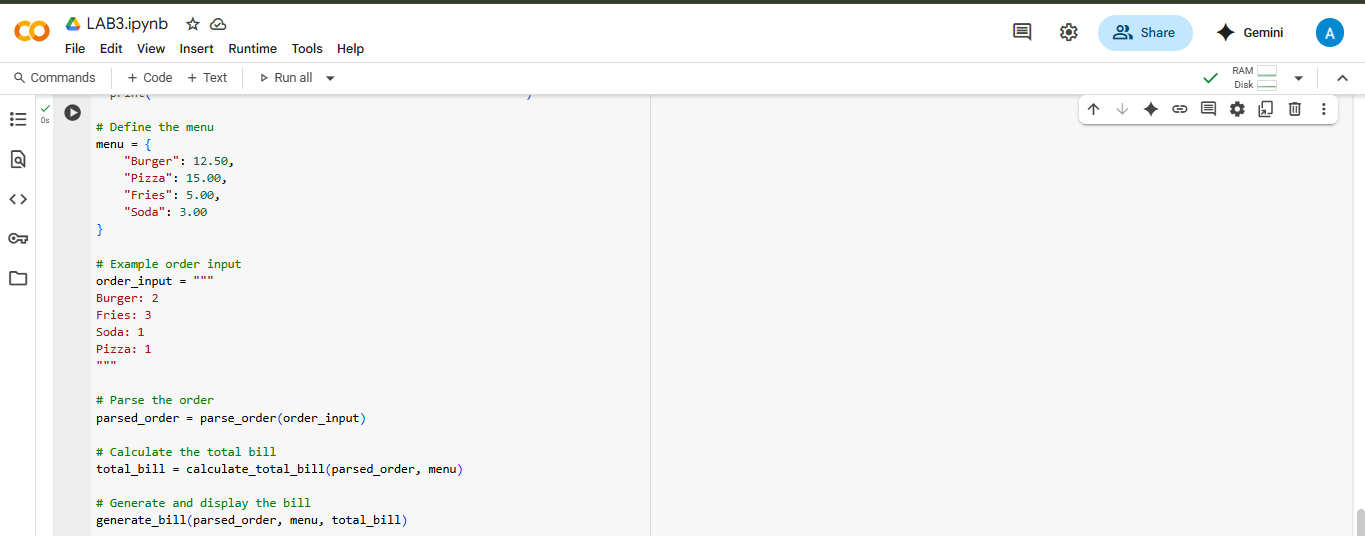
**Task – 4:**

write a python program to generate a hotel bill to input the items and quantity.

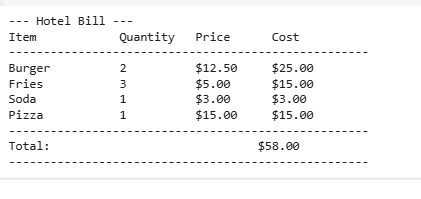
**Code:**







**Output:**



**Explanation:**

1. Parse\_order(order\_text) function:

* This function takes a string order\_text as input, which is expected to be a list of items and quantities, each on a new line (e.g., "Burger: 2").
* It splits the input string into individual lines and then splits each line by the colon (:) to separate the item name and the quantity.
* It tries to convert the quantity string into an integer. If successful, it stores the item name and quantity in a dictionary called order\_details.
* It includes basic error handling to warn you if a line is not in the expected format or if the quantity cannot be parsed as an integer.
* Finally, it returns the order\_details dictionary.

1. Calculate\_total\_bill(parsed\_order, menu) function:

* This function takes the parsed\_order dictionary (from the previous function) and the menu dictionary as input.
* It iterates through each item and quantity in the parsed\_order.
* For each item, it checks if the item exists in the menu.
* If the item is found in the menu, it calculates the cost for that item (quantity multiplied by the price from the menu) and adds it to the total\_amount.
* If an item is not found in the menu, it prints a warning message.
* It returns the calculated total\_amount.

3. Generate\_bill(parsed\_order, menu, total\_bill) function:

* This function takes the parsed\_order, menu, and total\_bill (from the previous function) as input.
* It prints a formatted header for the bill, including columns for 'Item', 'Quantity', 'Price', and 'Cost'.
* It then iterates through the items and quantities in the parsed\_order.
* For each item found in the menu, it retrieves the price and calculates the item's cost.
* It prints each item, its quantity, price, and calculated cost in a formatted row.
* Finally, it prints a formatted line showing the total bill amount at the bottom.

4. Defining the menu:

* A dictionary named menu is created to store the available items and their corresponding prices.

5. Example Usage:

* An example order\_input string is defined.
* The parse\_order function is called with order\_input to get the parsed\_order.
* The calculate\_total\_bill function is called with parsed\_order and menu to get the total\_bill.
* The generate\_bill function is called with parsed\_order, menu, and total\_bill to display the final formatted bill.

**Task – 5:**

write a program in python to convert Celsius to fahrenheit using functions

**Code:**



**Output:**



**Explanation:**

* def celsius\_to\_fahrenheit(celsius):: This line defines a function named celsius\_to\_fahrenheit that takes one argument, celsius.
* """Converts Celsius to Fahrenheit.""": This is a docstring, which explains what the function does. It's good practice to include these in your functions.
* fahrenheit = (celsius \* 9/5) + 32: This line performs the conversion using the standard formula: multiply Celsius by 9/5 and add 32. The result is stored in the fahrenheit variable.
* return fahrenheit: This line returns the calculated Fahrenheit value from the function. celsius\_temp = 25: This line sets a variable celsius\_temp to 25, which is the Celsius temperature we want to convert.
* fahrenheit\_temp = celsius\_to\_fahrenheit(celsius\_temp): This line calls the celsius\_to\_fahrenheit function with celsius\_temp (25) as input and stores the returned Fahrenheit value in fahrenheit\_temp.
* print(f"{celsius\_temp} degrees Celsius is equal to {fahrenheit\_temp} degrees Fahrenheit."): This line prints the result of the first conversion in a user-friendly format using an f-string.
* The last three lines repeat the process for a Celsius temperature of 0 degrees.