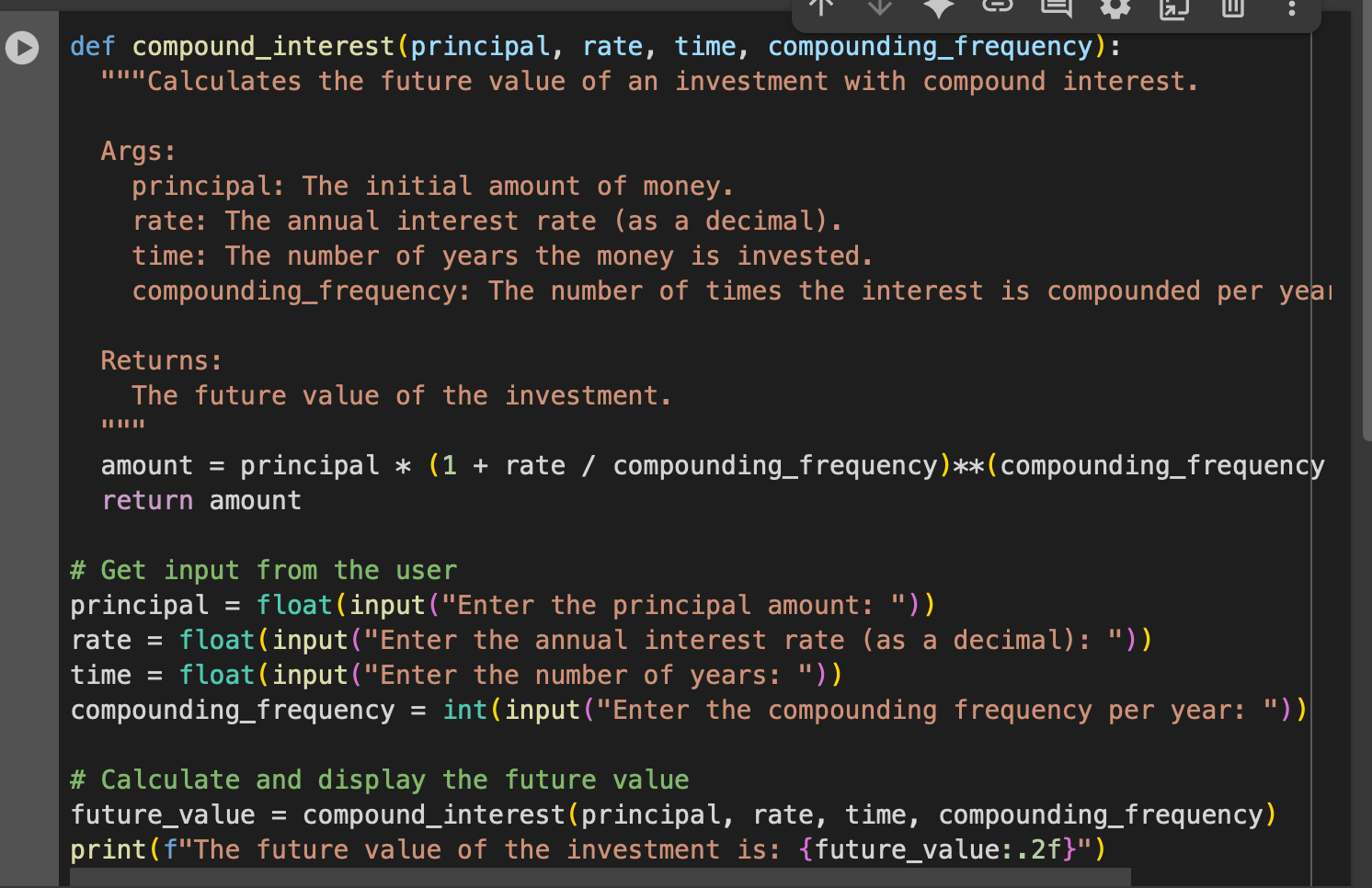
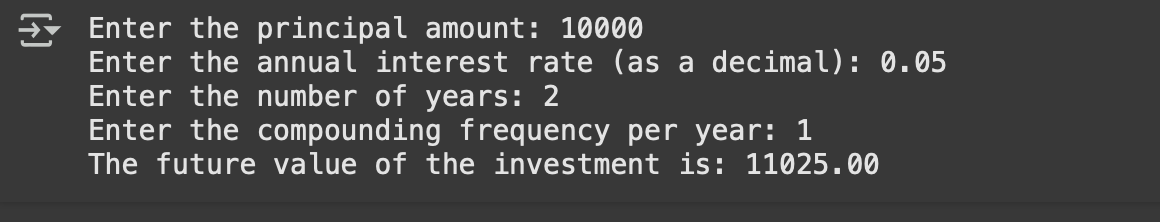
ASSIGNMENT 3.2  
(2403A52398 )

Task #1  
  
prompt : write a python function to calculate compound interest by taking principal, rate,no.of years,compounding frequency from the user

Code:  
  


Output:



Explanation :  
This code defines a Python function called compound\_interest that calculates the future value of an investment.

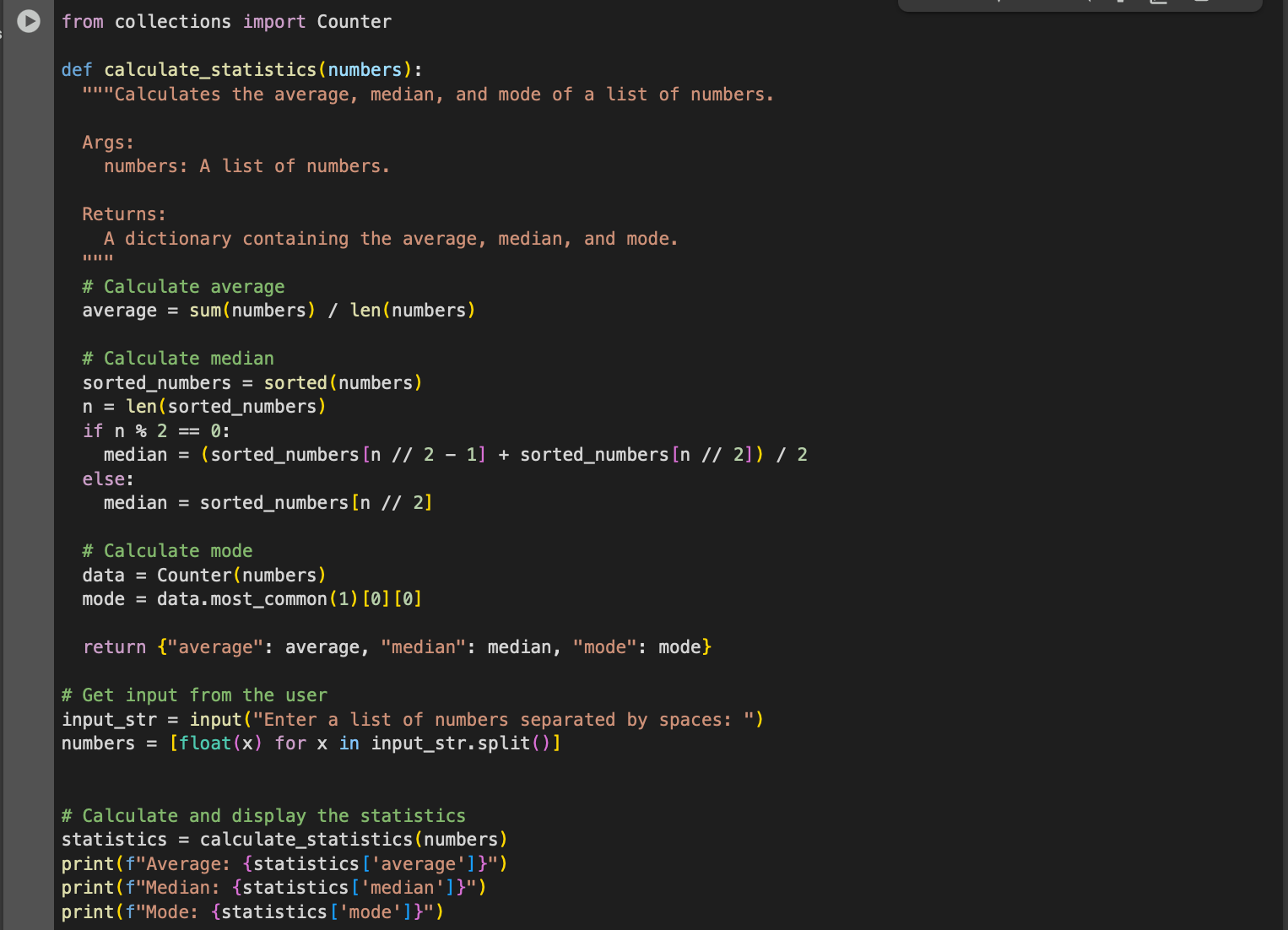
* The function takes four arguments: principal, rate, time, and compounding\_frequency.
* Inside the function, it uses the compound interest formula: amount = principal \* (1 + rate / compounding\_frequency)\*\*(compounding\_frequency \* time) to calculate the future value.
* It then returns the calculated amount.

After the function definition, the code prompts the user to enter the values for principal, rate, time, and compounding frequency using the input() function. These inputs are converted to the appropriate data types (float for principal, rate, and time, and int for compounding frequency).

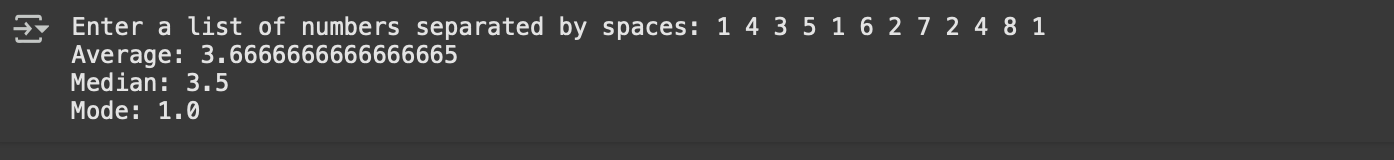
Finally, it calls the compound\_interest function with the user's input values and stores the returned future value in the future\_value variable. It then prints the future\_value formatted to two decimal places.

Task #2  
Prompt :

write a python function to calculate average , median , mode from the given list of numbers from the user , display average , median , mode in separate lines in output.

Code:  
  


Output :



Explanation:

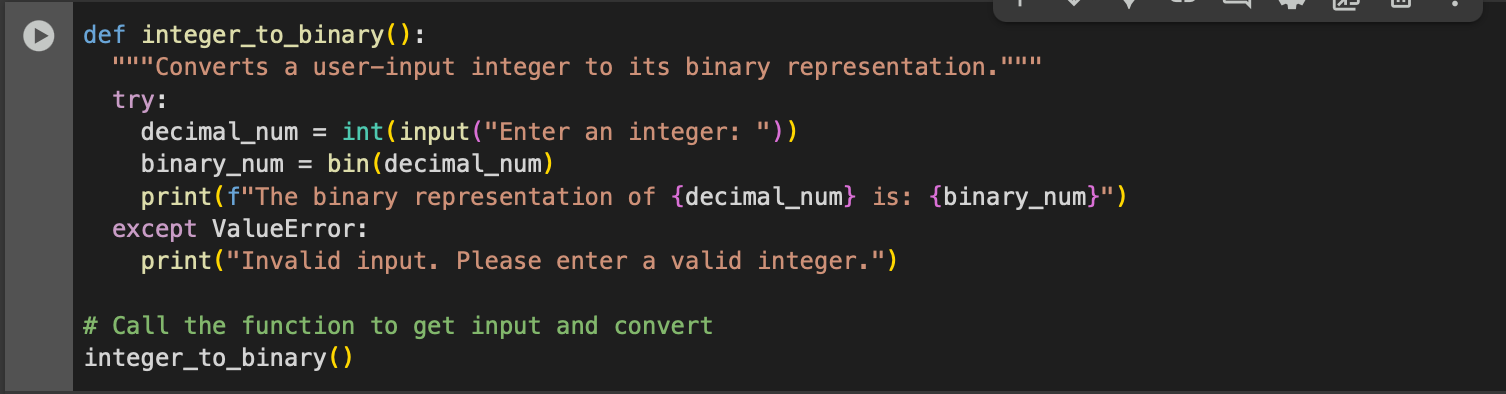
This code defines a Python function called calculate\_statisticsthat takes a list of numbers as input and returns a dictionary containing the average, median, and mode of those numbers.

Here's a breakdown of the code:

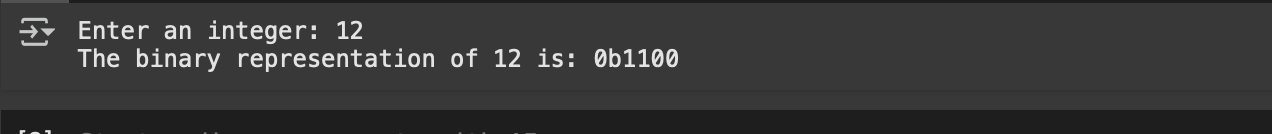
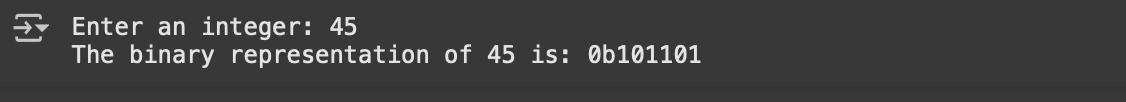
1. **Import** Counter**:** It imports the Counter class from the collections module, which is used to easily calculate the mode.
2. calculate\_statistics(numbers) **function:**
   1. **Average:** It calculates the average by summing all the numbers in the list and dividing by the number of elements in the list.
   2. **Median:** It calculates the median by first sorting the list of numbers. If the list has an even number of elements, the median is the average of the two middle elements. If the list has an odd number of elements, the median is the middle element.
   3. **Mode:** It calculates the mode by using Counter to count the occurrences of each number in the list. most\_common(1) returns a list of the most common element and its count, and [0][0] extracts the element itself (the mode).
   4. It returns a dictionary containing the calculated average, median, and mode.
3. **User Input:** The code prompts the user to enter a list of numbers separated by spaces using input(). It then splits the input string by spaces and converts each element to a float, storing them in the numbers list.
4. **Calculate and Display:** It calls the calculate\_statisticsfunction with the user-provided list of numbers and stores the result in the statistics variable. Finally, it prints the average, median, and mode from the statistics dictionary on separate lines.

Task#3

Prompt : write a function to convert user input integers to binary in python  
  
code:



Output:



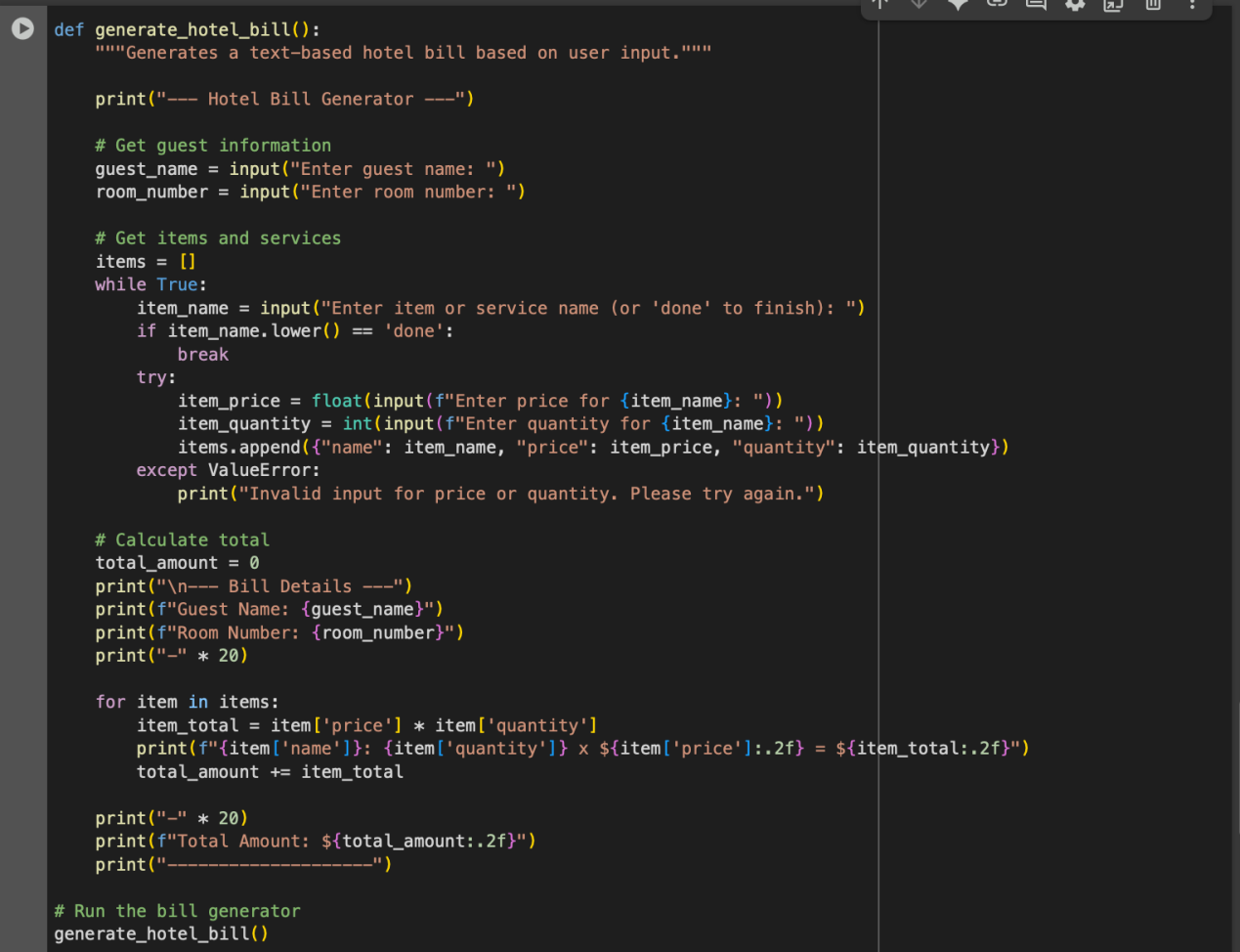
Explanation:

This code defines a Python function called integer\_to\_binary that converts an integer provided by the user into its binary representation.

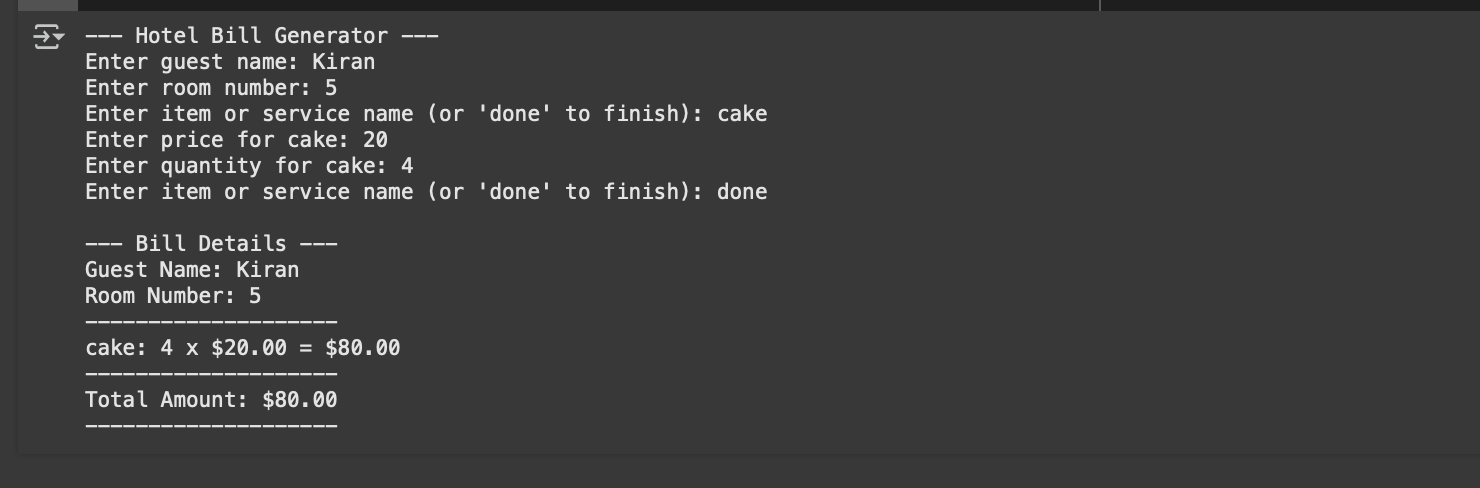
Here's a breakdown of the code:

1. integer\_to\_binary() **function:**
   1. It uses a try...except block to handle potential errors if the user enters something that is not a valid integer.
   2. Inside the try block:
      1. It prompts the user to "Enter an integer:" using the input() function and converts the input string to an integer using int(). This integer is stored in the decimal\_num variable.
      2. It uses the built-in Python function bin() to convert the decimal\_num to its binary representation. The result, which is a string starting with "0b" to indicate a binary number, is stored in the binary\_num variable.
      3. It prints a formatted string showing the original decimal number and its binary representation.
   3. Inside the except Value Error: block:
      1. If the user's input cannot be converted to an integer (e.g., they enter text), a Value Error occurs, and the code in this block is executed. It prints an error message "Invalid input. Please enter a valid integer."
2. integer\_to\_binary() **call:** Finally, the code calls the integer\_to\_binary() function to execute the process of getting user input and performing the conversion.

Task#4  
  
Prompt :  
create a python program to generate bill for an hotel based on customer requirements such as price , quantity of item , mention total at last.  
  
Code:



Output:

  
  
Explaination:

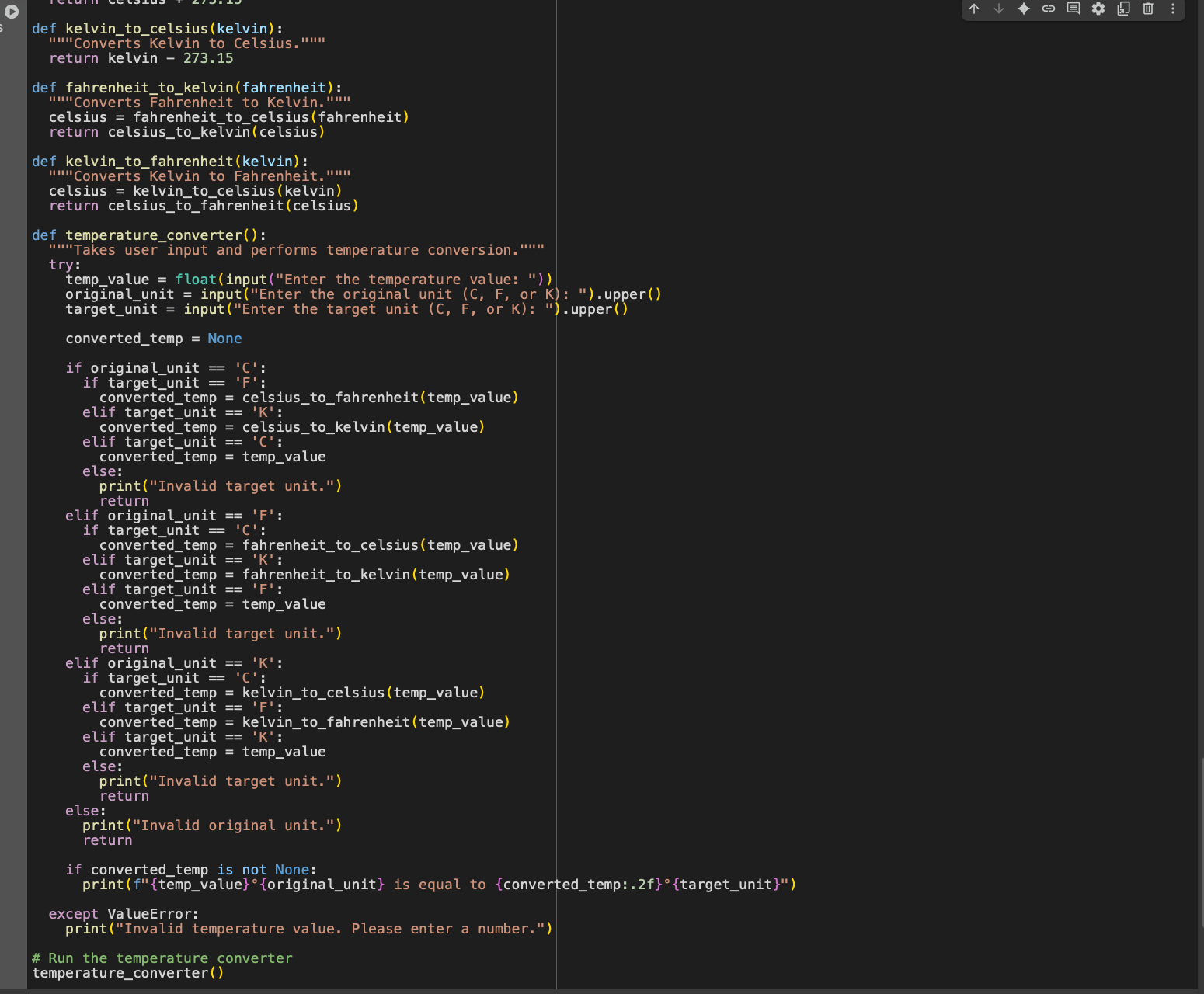
 code cell with the generate\_hotel\_bill function is a Python program that creates a simple text-based hotel bill. Here's how it works:

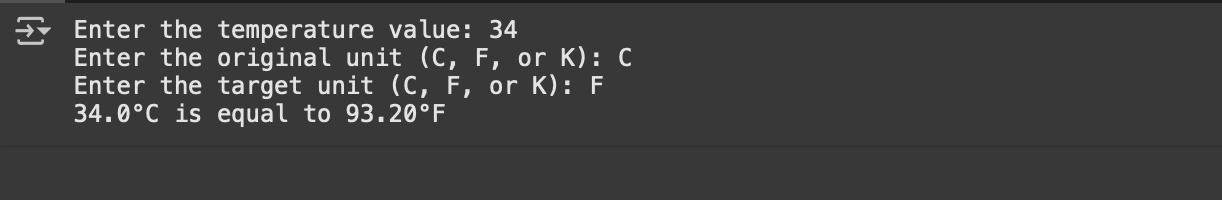
1. generate\_hotel\_bill() **function:** This function contains all the logic for generating the bill.
2. **Welcome Message:** It starts by printing a welcome message "--- Hotel Bill Generator ---".
3. **Get Guest Information:** It prompts the user to enter the guest's name and room number using the input() function and stores them in the guest\_name and room\_number variables.
4. **Get Items and Services:**
   1. An empty list called items is created to store the details of items and services.
   2. A while True loop is used to repeatedly ask for item details until the user types 'done'.
   3. Inside the loop, it prompts for the item/service name, price, and quantity.
   4. A try-except block is used to handle potential ValueError if the user enters non-numeric input for price or quantity.
   5. If the input is valid, a dictionary containing the item's name, price, and quantity is created and appended to the items list.
5. **Calculate Total and Display Bill:**
   1. A variable total\_amount is initialized to 0.
   2. It prints the "--- Bill Details ---" header, guest name, and room number.
   3. It then iterates through the items list. For each item, it calculates the item\_total (price \* quantity) and prints a formatted line showing the item name, quantity, price, and the total for that item. The total\_amount is updated by adding the item\_total.
   4. Finally, it prints a separator line and the Total Amount for the bill, formatted to two decimal places.
6. **Run the generator:** The line generate\_hotel\_bill() at the end calls the function to start the bill generation process.

In essence, the program collects guest and item details from the user, calculates the total cost of each item and the overall total, and then prints a formatted bill to the console.

Task#5

Prompt:  
  
create a program to make temperature conversions more efficient using functions , take one temperature from user input and convert it based on user requiremet

Code::  


Output :  
  


Explanation:

The code is designed to convert a temperature value from one unit (Celsius, Fahrenheit, or Kelvin) to another based on your input. Here's how it works:

1. **Conversion Functions:** The code starts with several small functions, each dedicated to a specific conversion:
   1. celsius\_to\_fahrenheit(celsius): Takes a Celsius temperature and returns the equivalent in Fahrenheit.
   2. fahrenheit\_to\_celsius(fahrenheit): Takes a Fahrenheit temperature and returns the equivalent in Celsius.
   3. celsius\_to\_kelvin(celsius): Takes a Celsius temperature and returns the equivalent in Kelvin.
   4. kelvin\_to\_celsius(kelvin): Takes a Kelvin temperature and returns the equivalent in Celsius.
   5. fahrenheit\_to\_kelvin(fahrenheit): Converts Fahrenheit to Kelvin by first converting to Celsius, then to Kelvin.
   6. kelvin\_to\_fahrenheit(kelvin): Converts Kelvin to Fahrenheit by first converting to Celsius, then to Fahrenheit.
2. temperature\_converter() **function:** This is the main function that interacts with the user:
   1. It uses a try-except block to handle potential errors if the user enters something that is not a number for the temperature value.
   2. Inside the try block, it prompts the user to enter the temperature value, the original unit (C, F, or K), and the target unit (C, F, or K). The .upper() method is used to convert the unit input to uppercase, making the comparison case-insensitive.
   3. It initializes a converted\_temp variable to None.
   4. A series of if-elif-else statements check the original\_unit and target\_unit to determine which conversion function to call.
   5. If the original and target units are the same, it just assigns the original temp\_value to converted\_temp.
   6. If the input units are invalid, it prints an error message and the function returns.
   7. If a valid conversion is performed, the result is stored in converted\_temp.
   8. Finally, if converted\_temp is not None (meaning a valid conversion happened), it prints the original temperature and units and the converted temperature and units, formatted to two decimal places.
   9. If a ValueError occurs (e.g., non-numeric input for temperature), the except block catches it and prints an error message.
3. **Running the function:** The last line temperature\_converter() calls the main function to start the program and prompt the user for input.