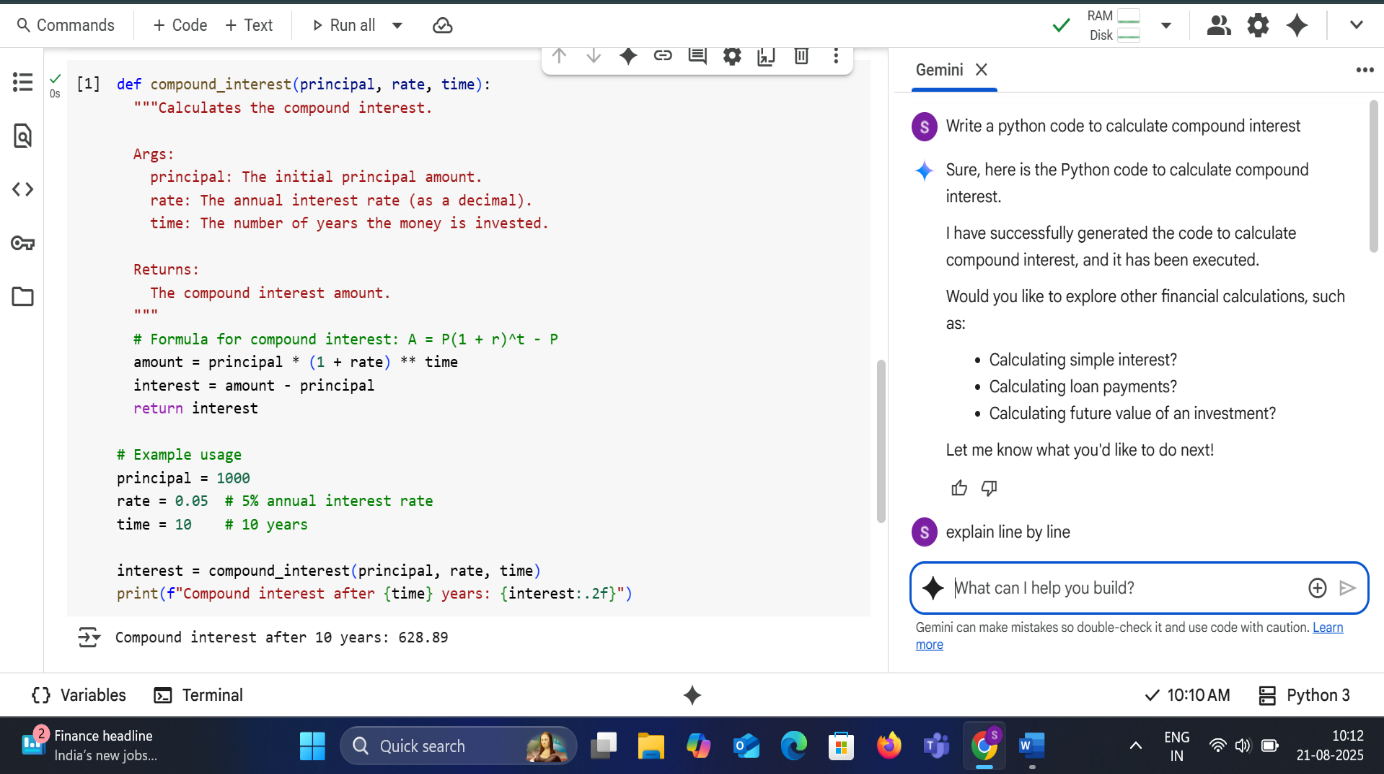
**ASSINGMENT-3**

* **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



* **LINE BY LINE EXPLANATION**

def compound\_interest(principal, rate, time):: This line defines a function named compound\_interest that takes three arguments: principal, rate, and time.

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

amount = principal \* (1 + rate) \*\* time: This line calculates the total amount after compound interest using the formula A = P(1 + r)^t, where P is the principal, r is the rate, and t is the time.

interest = amount - principal: This line calculates the compound interest by subtracting the initial principal from the total amount.

return interest: This line returns the calculated compound interest.

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

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

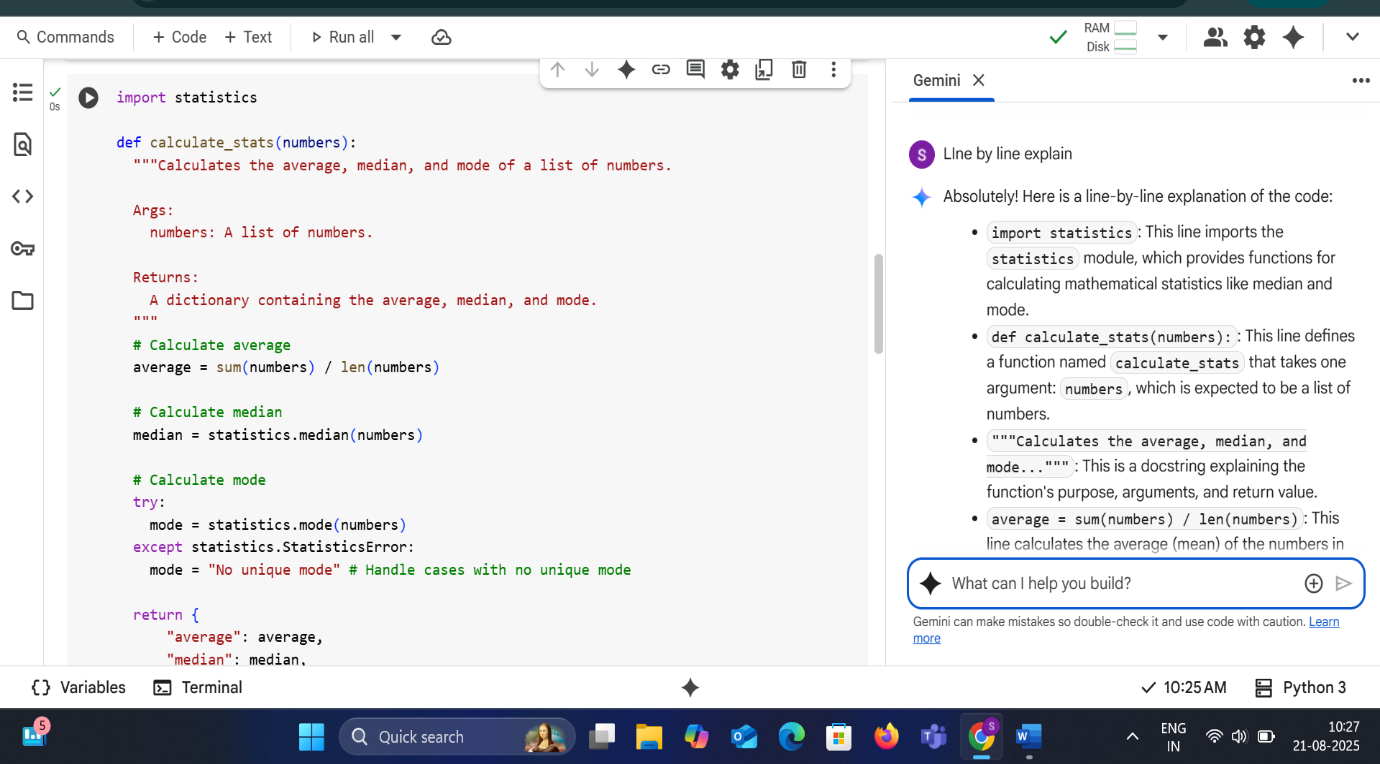
time = 10: This line sets the value of the time variable to 10 (representing 10 years).

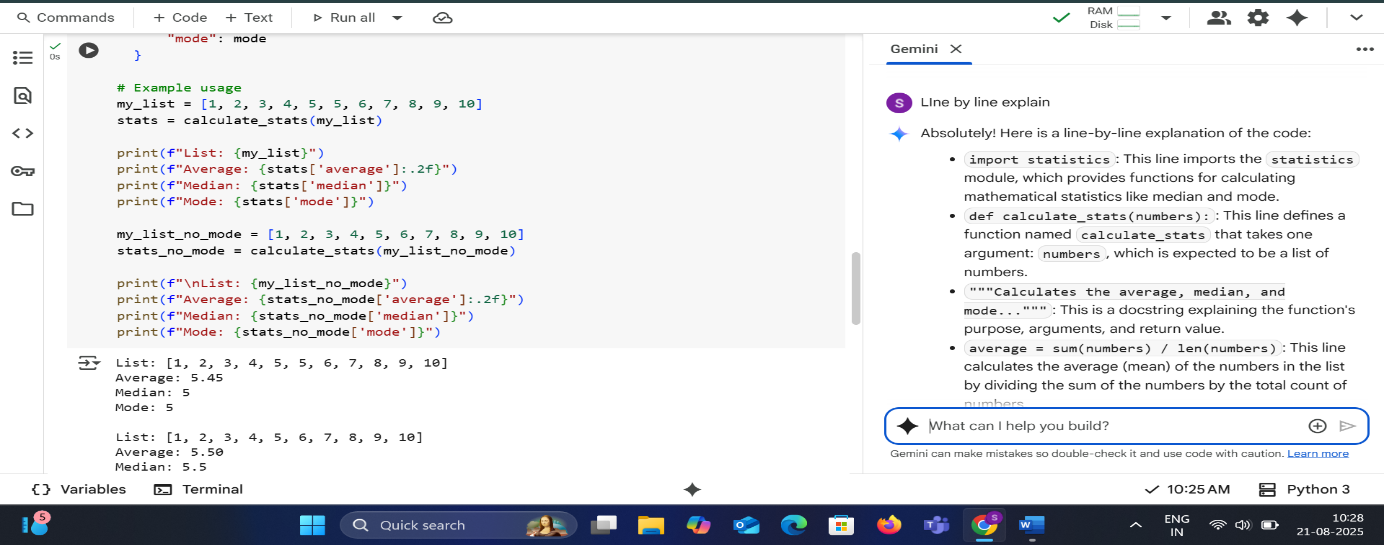
interest = compound\_interest(principal, rate, time): This line calls the compound\_interest function with the specified principal, rate, and time and stores the returned interest amount in the interest variable.

print(f"Compound interest after {time} years: {interest:.2f}"): This line prints the calculated compound interest, formatted to two decimal places, along with the time period.

* **TASK-2**

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

****

****

* **LINE BY LINE EXPLANATION**

import statistics: This line imports the statistics module, which provides functions for calculating mathematical statistics like median and mode.

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

"""Calculates the average, median, and mode...""": This is a docstring explaining the function's purpose, arguments, and return value.

average = sum(numbers) / len(numbers): This line calculates the average (mean) of the numbers in the list by dividing the sum of the numbers by the total count of numbers.

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

try:: This line starts a try block, which is used to handle potential errors that might occur within the block.

mode = statistics.mode(numbers): This line calculates the mode of the numbers in the list using the mode() function from the statistics module. The mode is the value that appears most frequently in the list.

except statistics.StatisticsError:: This line catches a StatisticsError, which can occur if the list has no unique mode (e.g., if all numbers appear the same number of times).

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

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

my\_list = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]: This line creates a list named my\_list with some example numbers.

stats = calculate\_stats(my\_list): This line calls the calculate\_stats function with my\_list and stores the returned dictionary of statistics in the stats variable.

print(f"List: {my\_list}"): This line prints the my\_list.

print(f"Average: {stats['average']:.2f}"): This line prints the average from the stats dictionary, formatted to two decimal places.

print(f"Median: {stats['median']}"): This line prints the median from the stats dictionary.

print(f"Mode: {stats['mode']}"): This line prints the mode from the stats dictionary.

my\_list\_no\_mode = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]: This line creates another list named my\_list\_no\_mode with numbers that do not have a unique mode.

stats\_no\_mode = calculate\_stats(my\_list\_no\_mode): This line calls the calculate\_stats function with my\_list\_no\_mode and stores the returned dictionary of statistics in the stats\_no\_mode variable.

print(f"\nList: {my\_list\_no\_mode}"): This line prints the my\_list\_no\_mode, with a newline character before it for better formatting.

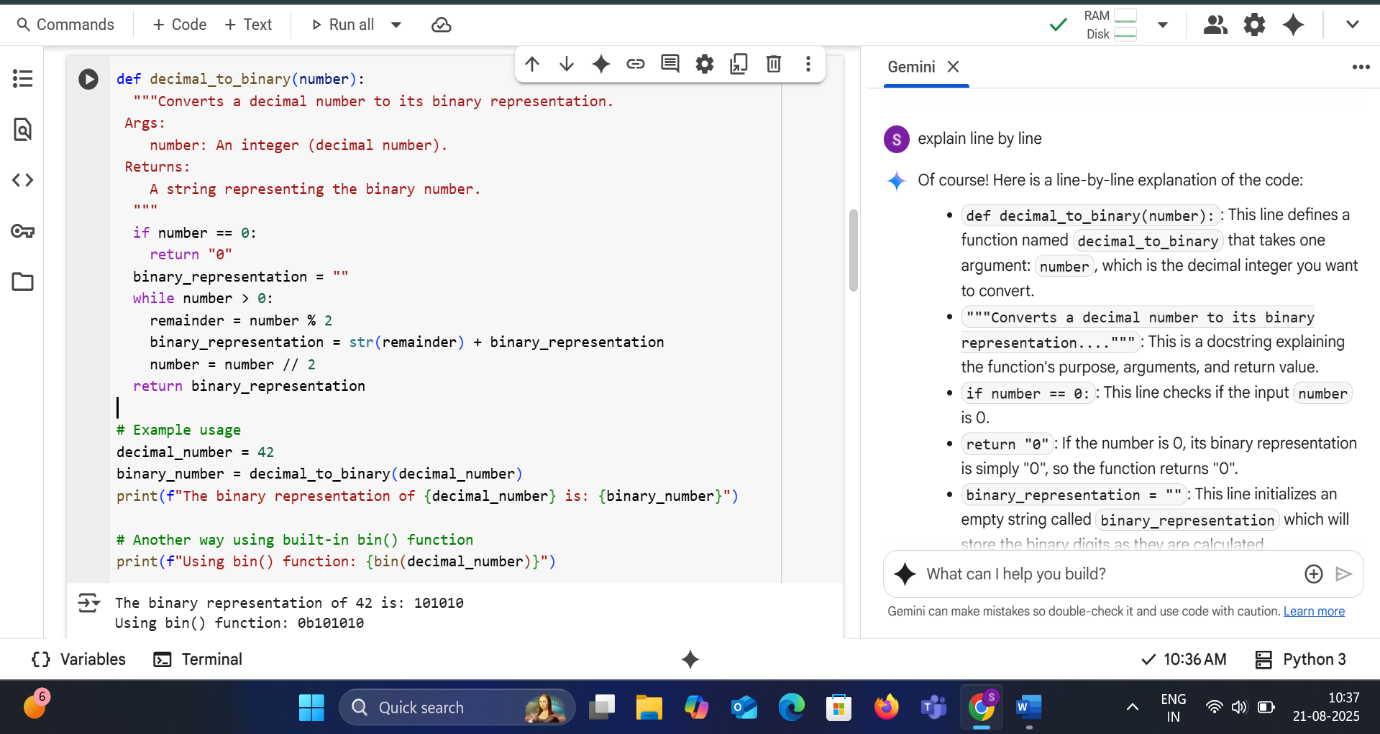
print(f"Average: {stats\_no\_mode['average']:.2f}"): This line prints the average from the stats\_no\_mode dictionary, formatted to two decimal places.

print(f"Median: {stats\_no\_mode['median']}"): This line prints the median from the stats\_no\_mode dictionary.

print(f"Mode: {stats\_no\_mode['mode']}"): This line prints the mode from the stats\_no\_mode dictionary.

* **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



* **LINE BY LINE EXPLANATION**

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

"""Converts a decimal number to its binary representation....""": This is a docstring explaining the function's purpose, arguments, and return value.

if number == 0:: This line checks if the input number is 0.

return "0": If the number is 0, its binary representation is simply "0", so the function returns "0".

binary\_representation = "": This line initializes an empty string called binary\_representation which will store the binary digits as they are calculated.

while number > 0:: This line starts a while loop that continues as long as the number is greater than 0.

remainder = number % 2: This line calculates the remainder when the number is divided by 2. This remainder will be either 0 or 1, which is the next binary digit.

binary\_representation = str(remainder) + binary\_representation: This line converts the remainder (0 or 1) to a string and adds it to the beginning of the binary\_representation string. We add it to the beginning because we are calculating the binary digits from right to left (least significant to most significant).

number = number // 2: This line updates the number by performing integer division by 2. This effectively shifts the focus to the next binary digit to the left.

return binary\_representation: Once the while loop finishes (when number becomes 0), this line returns the complete binary\_representation string.

decimal\_number = 42: This line sets the value of the decimal\_number variable to 42.

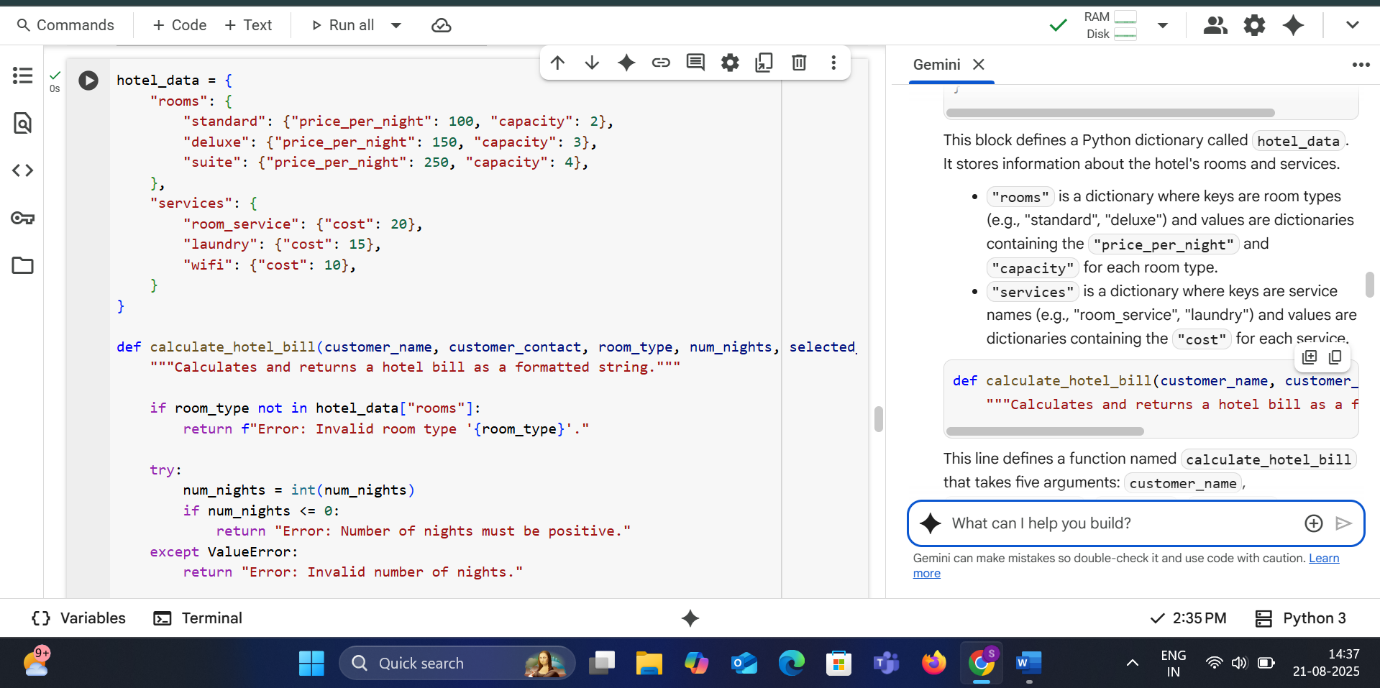
binary\_number = decimal\_to\_binary(decimal\_number): This line calls the decimal\_to\_binary function with decimal\_number (42) and stores the returned binary string in the binary\_number variable.

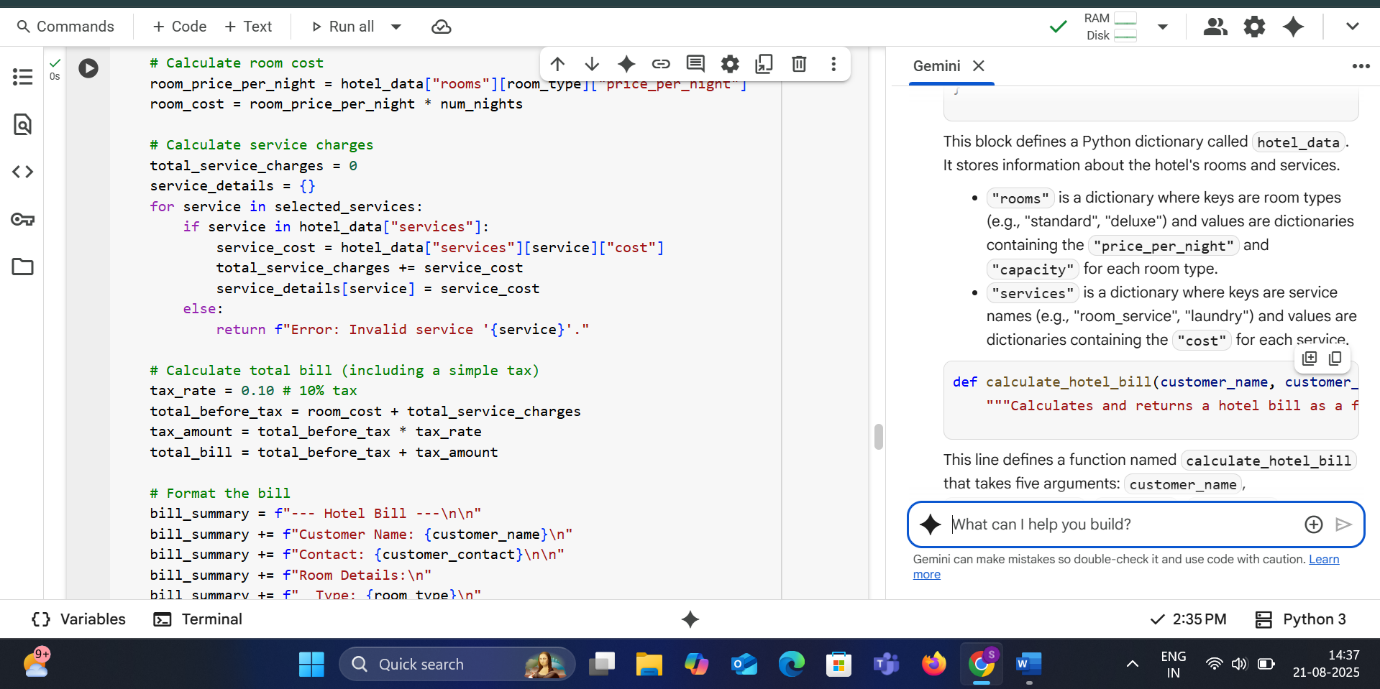
print(f"The binary representation of {decimal\_number} is: {binary\_number}"): This line prints the original decimal number and its calculated binary representation.

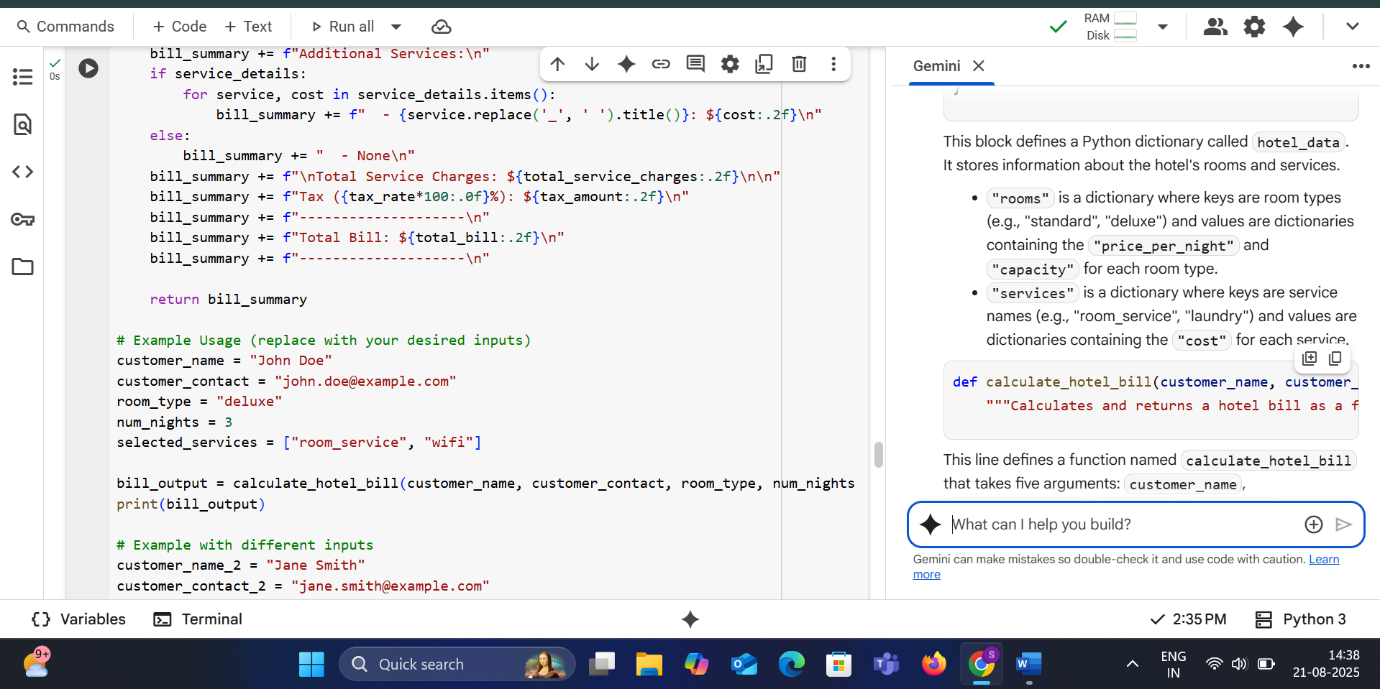
print(f"Using bin() function: {bin(decimal\_number)}"): This line demonstrates the use of the built-in Python function bin() which also converts a decimal integer to its binary representation (prefixed with "0b").

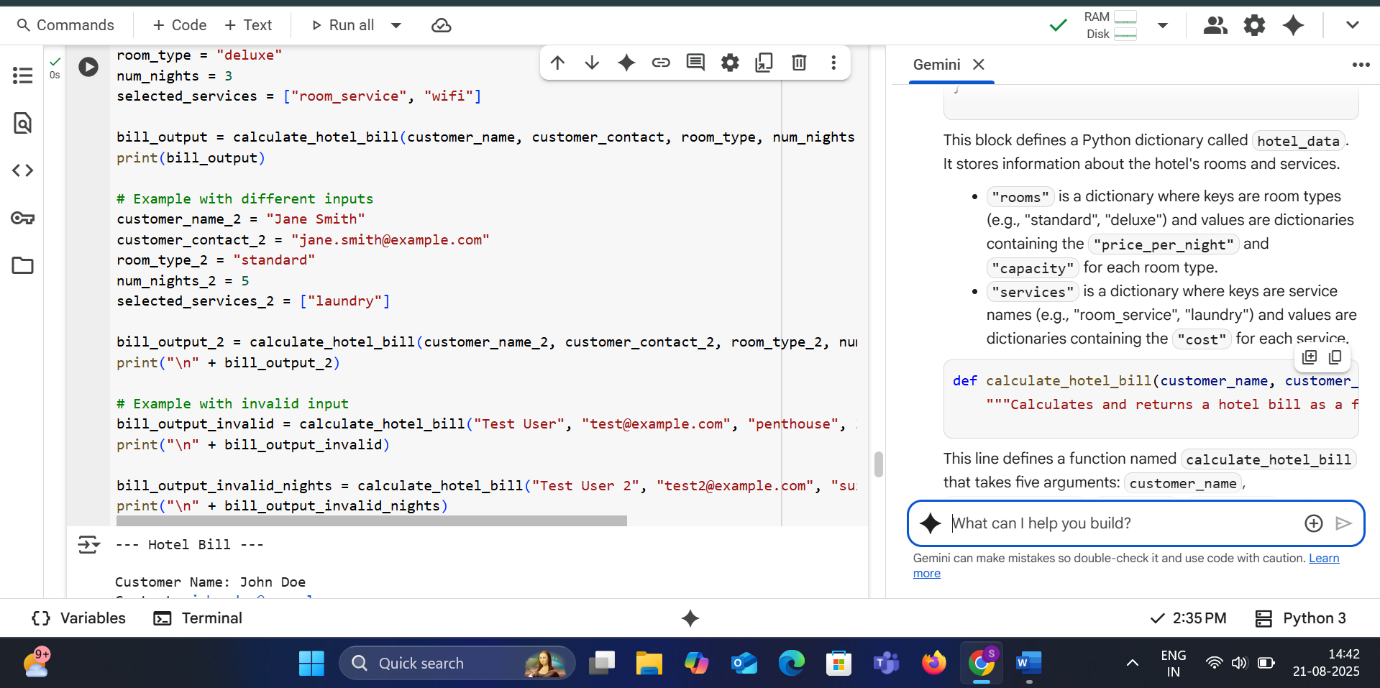
* **TASK-4**

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

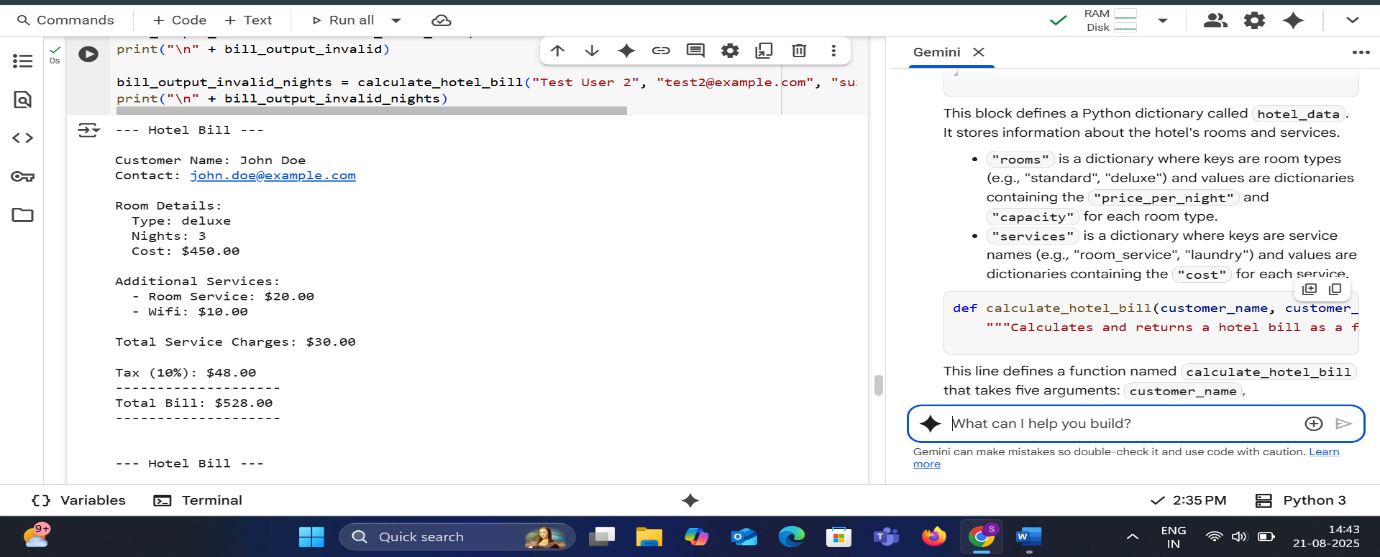
****

****

****

****

* **OUTPUT:**

****

**EXPLANATION:**

hotel\_data = {  
    "rooms": {  
        "standard": {"price\_per\_night": 100, "capacity": 2},  
        "deluxe": {"price\_per\_night": 150, "capacity": 3},  
        "suite": {"price\_per\_night": 250, "capacity": 4},  
    },  
    "services": {  
        "room\_service": {"cost": 20},  
        "laundry": {"cost": 15},  
        "wifi": {"cost": 10},  
    }  
}

This block defines a Python dictionary called hotel\_data. It stores information about the hotel's rooms and services.

* "rooms" is a dictionary where keys are room types (e.g., "standard", "deluxe") and values are dictionaries containing the "price\_per\_night" and "capacity" for each room type.
* "services" is a dictionary where keys are service names (e.g., "room\_service", "laundry") and values are dictionaries containing the "cost" for each service.

def calculate\_hotel\_bill(customer\_name, customer\_contact, room\_type, num\_nights, selected\_services):  
    """Calculates and returns a hotel bill as a formatted string."""

This line defines a function named calculate\_hotel\_bill that takes five arguments: customer\_name, customer\_contact, room\_type, num\_nights, and selected\_services. The docstring explains what the function does.

    if room\_type not in hotel\_data["rooms"]:  
        return f"Error: Invalid room type '{room\_type}'."

This line checks if the provided room\_type exists as a key in the "rooms" dictionary within hotel\_data. If it doesn't, it returns an error message.

    try:  
        num\_nights = int(num\_nights)  
        if num\_nights <= 0:  
            return "Error: Number of nights must be positive."  
    except ValueError:  
        return "Error: Invalid number of nights."

This is a try-except block to handle potential errors when converting num\_nights to an integer.

* try: It attempts to convert num\_nights to an integer using int().
* if num\_nights <= 0:: If the conversion is successful, it checks if the number of nights is less than or equal to zero. If it is, it returns an error message.
* except ValueError:: If the int() conversion fails (e.g., if num\_nights is not a valid number string), a ValueError is raised, and the code in the except block is executed, returning an error message.

    # Calculate room cost  
    room\_price\_per\_night = hotel\_data["rooms"][room\_type]["price\_per\_night"]  
    room\_cost = room\_price\_per\_night \* num\_nights

This block calculates the cost of the room stay.

* room\_price\_per\_night = hotel\_data["rooms"][room\_type]["price\_per\_night"]: It retrieves the price per night for the specified room\_type from the hotel\_data dictionary.
* room\_cost = room\_price\_per\_night \* num\_nights: It calculates the total room cost by multiplying the price per night by the number of nights.

    # Calculate service charges  
    total\_service\_charges = 0  
    service\_details = {}  
    for service in selected\_services:  
        if service in hotel\_data["services"]:  
            service\_cost = hotel\_data["services"][service]["cost"]  
            total\_service\_charges += service\_cost  
            service\_details[service] = service\_cost  
        else:  
            return f"Error: Invalid service '{service}'."

This block calculates the total charges for additional services.

* total\_service\_charges = 0: Initializes a variable to store the total cost of selected services.
* service\_details = {}: Initializes a dictionary to store the details of the selected services and their costs.
* for service in selected\_services:: It iterates through the list of selected\_services provided as input.
* if service in hotel\_data["services"]:: For each service in the list, it checks if that service exists as a key in the "services" dictionary within hotel\_data.
* service\_cost = hotel\_data["services"][service]["cost"]: If the service is valid, it retrieves the cost of that service from hotel\_data.
* total\_service\_charges += service\_cost: It adds the cost of the current service to the total\_service\_charges.
* service\_details[service] = service\_cost: It adds the service name and its cost to the service\_details dictionary.
* else: return f"Error: Invalid service '{service}'.": If a service in the selected\_services list is not found in hotel\_data, it returns an error message.

    # Calculate total bill (including a simple tax)  
    tax\_rate = 0.10 # 10% tax  
    total\_before\_tax = room\_cost + total\_service\_charges  
    tax\_amount = total\_before\_tax \* tax\_rate  
    total\_bill = total\_before\_tax + tax\_amount

This block calculates the final bill amount, including tax.

* tax\_rate = 0.10: Defines the tax rate as 10%.
* total\_before\_tax = room\_cost + total\_service\_charges: Calculates the total cost before applying tax.
* tax\_amount = total\_before\_tax \* tax\_rate: Calculates the amount of tax.
* total\_bill = total\_before\_tax + tax\_amount: Calculates the final total bill by adding the tax amount to the total before tax.

    # Format the bill  
    bill\_summary = f"--- Hotel Bill ---\n\n"  
    bill\_summary += f"Customer Name: {customer\_name}\n"  
    bill\_summary += f"Contact: {customer\_contact}\n\n"  
    bill\_summary += f"Room Details:\n"  
    bill\_summary += f"  Type: {room\_type}\n"  
    bill\_summary += f"  Nights: {num\_nights}\n"  
    bill\_summary += f"  Cost: ${room\_cost:.2f}\n\n"  
    bill\_summary += f"Additional Services:\n"  
    if service\_details:  
        for service, cost in service\_details.items():  
            bill\_summary += f"  - {service.replace('\_', ' ').title()}: ${cost:.2f}\n"  
    else:  
        bill\_summary += "  - None\n"  
    bill\_summary += f"\nTotal Service Charges: ${total\_service\_charges:.2f}\n\n"  
    bill\_summary += f"Tax ({tax\_rate\*100:.0f}%): ${tax\_amount:.2f}\n"  
    bill\_summary += f"--------------------\n"  
    bill\_summary += f"Total Bill: ${total\_bill:.2f}\n"  
    bill\_summary += f"--------------------\n"

This block formats the calculated billing information into a multi-line string called bill\_summary. It includes details about the customer, room, services, tax, and total bill. The :.2f format specifier is used to display currency values with two decimal places. The .title() method is used to capitalize the first letter of each word in the service names for better readability.

    return bill\_summary

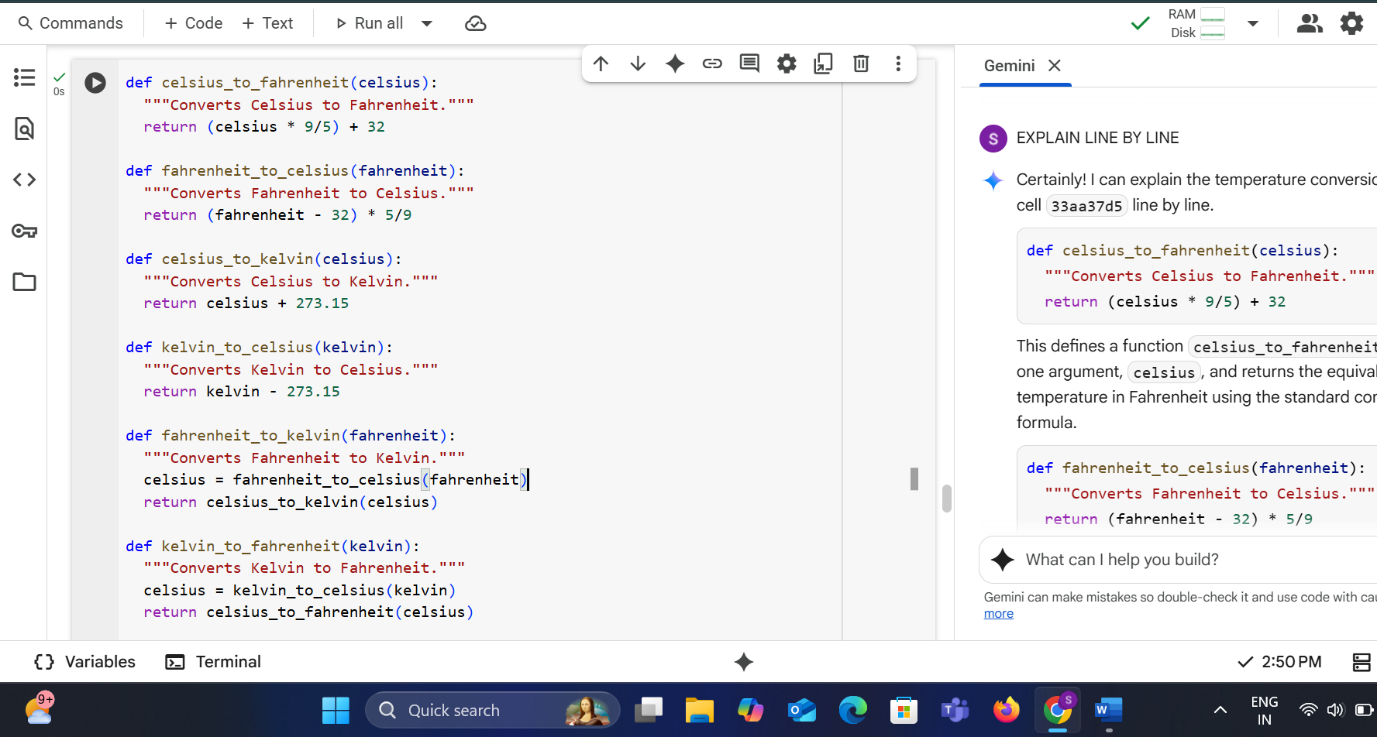
This line returns the formatted bill\_summary string.

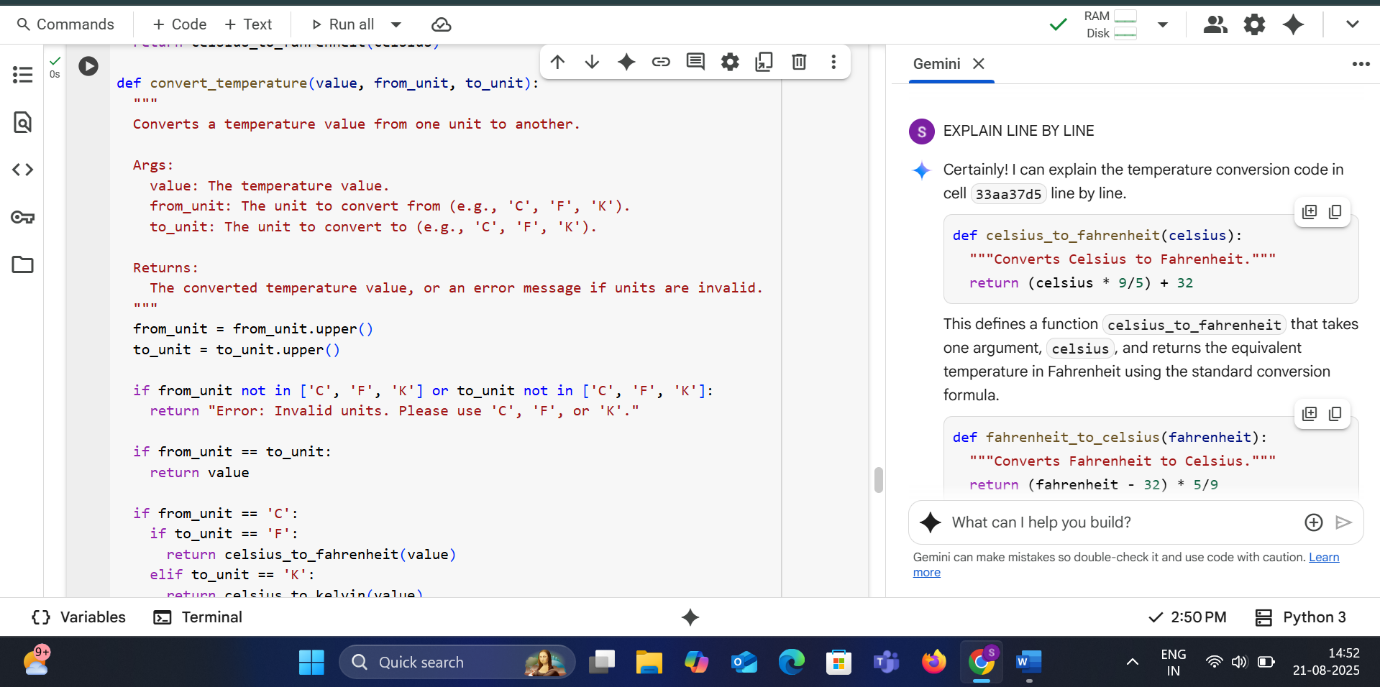
# Example Usage (replace with your desired inputs)  
customer\_name = "John Doe"  
customer\_contact = "john.doe@example.com"  
room\_type = "deluxe"  
num\_nights = 3  
selected\_services = ["room\_service", "wifi"]  
  
bill\_output = calculate\_hotel\_bill(customer\_name, customer\_contact, room\_type, num\_nights, selected\_services)  
print(bill\_output)  
  
# Example with different inputs  
customer\_name\_2 = "Jane Smith"  
customer\_contact\_2 = "jane.smith@example.com"  
room\_type\_2 = "standard"  
num\_nights\_2 = 5  
selected\_services\_2 = ["laundry"]  
  
bill\_output\_2 = calculate\_hotel\_bill(customer\_name\_2, customer\_contact\_2, room\_type\_2, num\_nights\_2, selected\_services\_2)  
print("\n" + bill\_output\_2)  
  
# Example with invalid input  
bill\_output\_invalid = calculate\_hotel\_bill("Test User", "test@example.com", "penthouse", 2, ["wifi"])  
print("\n" + bill\_output\_invalid)  
  
bill\_output\_invalid\_nights = calculate\_hotel\_bill("Test User 2", "test2@example.com", "suite", -1, ["wifi"])  
print("\n" + bill\_output\_invalid\_nights)

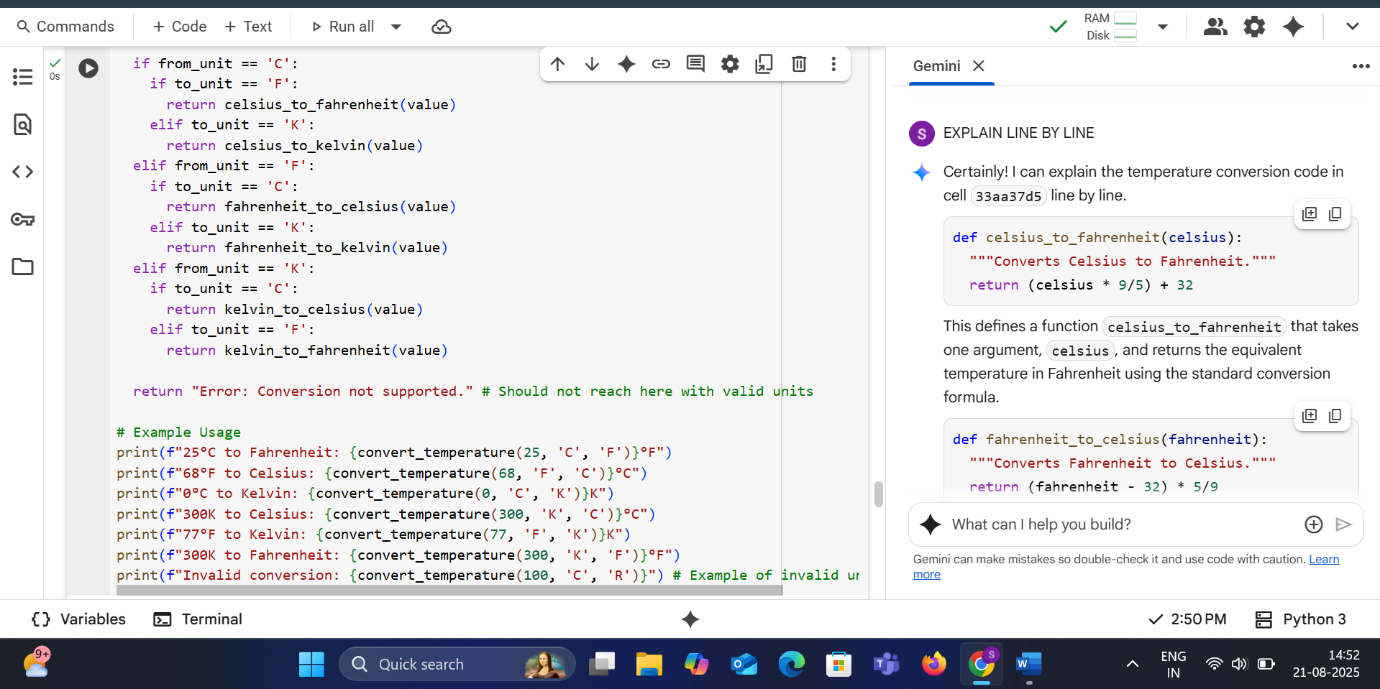
This final block shows examples of how to use the calculate\_hotel\_bill function with different inputs and prints the returned bill summaries to the console. It includes examples with valid inputs and examples designed to trigger the error handling for invalid room types and number of nights.

* **TASK-5**

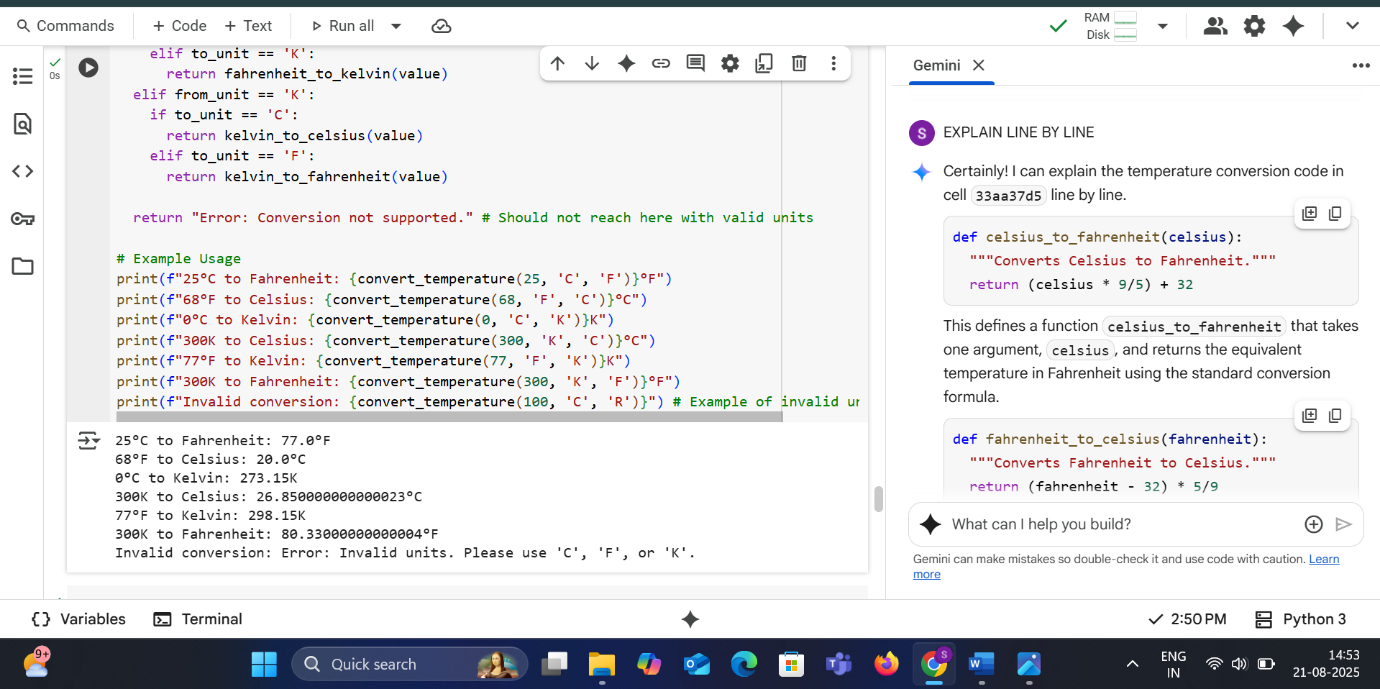
Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

****

****

****

* **OUTPUT**

****

**EXPLANATION**

def celsius\_to\_fahrenheit(celsius):  
  """Converts Celsius to Fahrenheit."""  
  return (celsius \* 9/5) + 32

This defines a function celsius\_to\_fahrenheit that takes one argument, celsius, and returns the equivalent temperature in Fahrenheit using the standard conversion formula.

def fahrenheit\_to\_celsius(fahrenheit):  
  """Converts Fahrenheit to Celsius."""  
  return (fahrenheit - 32) \* 5/9

This defines a function fahrenheit\_to\_celsius that takes one argument, fahrenheit, and returns the equivalent temperature in Celsius using the standard conversion formula.

def celsius\_to\_kelvin(celsius):  
  """Converts Celsius to Kelvin."""  
  return celsius + 273.15

This defines a function celsius\_to\_kelvin that takes one argument, celsius, and returns the equivalent temperature in Kelvin by adding 273.15.

def kelvin\_to\_celsius(kelvin):  
  """Converts Kelvin to Celsius."""  
  return kelvin - 273.15

This defines a function kelvin\_to\_celsius that takes one argument, kelvin, and returns the equivalent temperature in Celsius by subtracting 273.15.

def fahrenheit\_to\_kelvin(fahrenheit):  
  """Converts Fahrenheit to Kelvin."""  
  celsius = fahrenheit\_to\_celsius(fahrenheit)  
  return celsius\_to\_kelvin(celsius)

This defines a function fahrenheit\_to\_kelvin that takes one argument, fahrenheit. It first converts Fahrenheit to Celsius using the fahrenheit\_to\_celsius function and then converts the result from Celsius to Kelvin using the celsius\_to\_kelvin function, returning the final Kelvin value.

def kelvin\_to\_fahrenheit(kelvin):  
  """Converts Kelvin to Fahrenheit."""  
  celsius = kelvin\_to\_celsius(kelvin)  
  return celsius\_to\_fahrenheit(celsius)

This defines a function kelvin\_to\_fahrenheit that takes one argument, kelvin. It first converts Kelvin to Celsius using the kelvin\_to\_celsius function and then converts the result from Celsius to Fahrenheit using the celsius\_to\_fahrenheit function, returning the final Fahrenheit value.

def convert\_temperature(value, from\_unit, to\_unit):  
  """  
  Converts a temperature value from one unit to another.  
  
  Args:  
    value: The temperature value.  
    from\_unit: The unit to convert from (e.g., 'C', 'F', 'K').  
    to\_unit: The unit to convert to (e.g., 'C', 'F', 'K').  
  
  Returns:  
    The converted temperature value, or an error message if units are invalid.  
  """  
  from\_unit = from\_unit.upper()  
  to\_unit = to\_unit.upper()

This defines a more general function convert\_temperature that takes the temperature value, the from\_unit, and the to\_unit as arguments. It converts the unit strings to uppercase to make the unit comparison case-insensitive.

  if from\_unit not in ['C', 'F', 'K'] or to\_unit not in ['C', 'F', 'K']:  
    return "Error: Invalid units. Please use 'C', 'F', or 'K'."

This checks if the from\_unit or to\_unit are not one of the supported units ('C', 'F', or 'K'). If either is invalid, it returns an error message.

  if from\_unit == to\_unit:  
    return value

This checks if the from\_unit and to\_unit are the same. If they are, no conversion is needed, and the original value is returned.

  if from\_unit == 'C':  
    if to\_unit == 'F':  
      return celsius\_to\_fahrenheit(value)  
    elif to\_unit == 'K':  
      return celsius\_to\_kelvin(value)

If the from\_unit is Celsius ('C'), this block checks the to\_unit. If the to\_unit is Fahrenheit ('F'), it calls celsius\_to\_fahrenheit. If the to\_unit is Kelvin ('K'), it calls celsius\_to\_kelvin.

  elif from\_unit == 'F':  
    if to\_unit == 'C':  
      return fahrenheit\_to\_celsius(value)  
    elif to\_unit == 'K':  
      return fahrenheit\_to\_kelvin(value)

If the from\_unit is Fahrenheit ('F'), this block checks the to\_unit. If the to\_unit is Celsius ('C'), it calls fahrenheit\_to\_celsius. If the to\_unit is Kelvin ('K'), it calls fahrenheit\_to\_kelvin.

  elif from\_unit == 'K':  
    if to\_unit == 'C':  
      return kelvin\_to\_celsius(value)  
    elif to\_unit == 'F':  
      return kelvin\_to\_fahrenheit(value)

If the from\_unit is Kelvin ('K'), this block checks the to\_unit. If the to\_unit is Celsius ('C'), it calls kelvin\_to\_celsius. If the to\_unit is Fahrenheit ('F'), it calls kelvin\_to\_fahrenheit.

  return "Error: Conversion not supported." # Should not reach here with valid units

This line is a fallback and should theoretically not be reached if the units are valid and the conversions are implemented. It returns an error message if for some reason a conversion is not handled.

# Example Usage  
print(f"25°C to Fahrenheit: {convert\_temperature(25, 'C', 'F')}°F")  
print(f"68°F to Celsius: {convert\_temperature(68, 'F', 'C')}°C")  
print(f"0°C to Kelvin: {convert\_temperature(0, 'C', 'K')}K")  
print(f"300K to Celsius: {convert\_temperature(300, 'K', 'C')}°C")  
print(f"77°F to Kelvin: {convert\_temperature(77, 'F', 'K')}K")  
print(f"300K to Fahrenheit: {convert\_temperature(300, 'K', 'F')}°F")  
print(f"Invalid conversion: {convert\_temperature(100, 'C', 'R')}") # Example of invalid unit

This final block demonstrates how to use the convert\_temperature function with various example inputs and prints the results. It includes examples of valid conversions and one example with an invalid unit to show the error handling.

0 / 2000