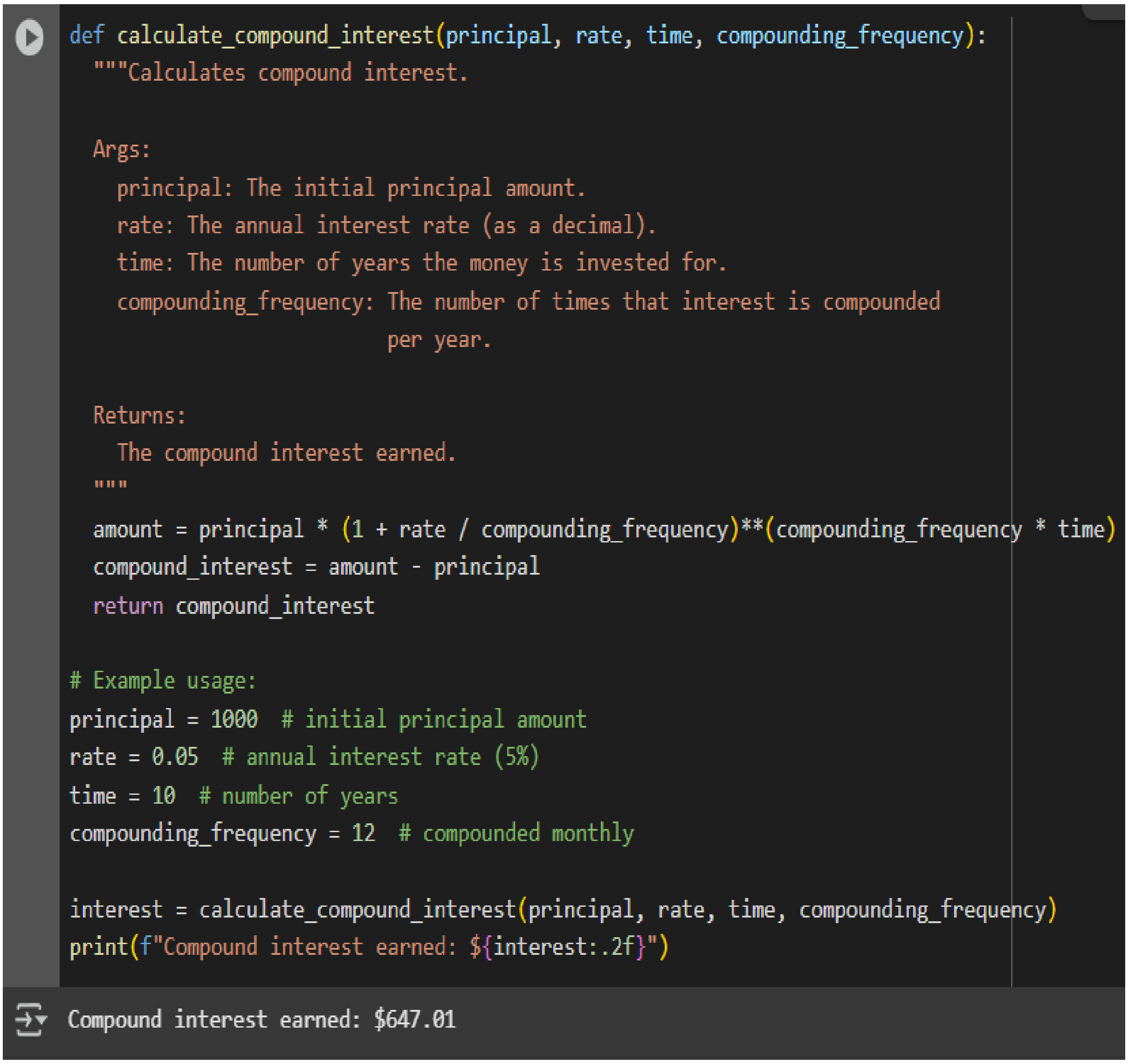
ASSIGNMENT 3

SAI CHARAN

2403A52124

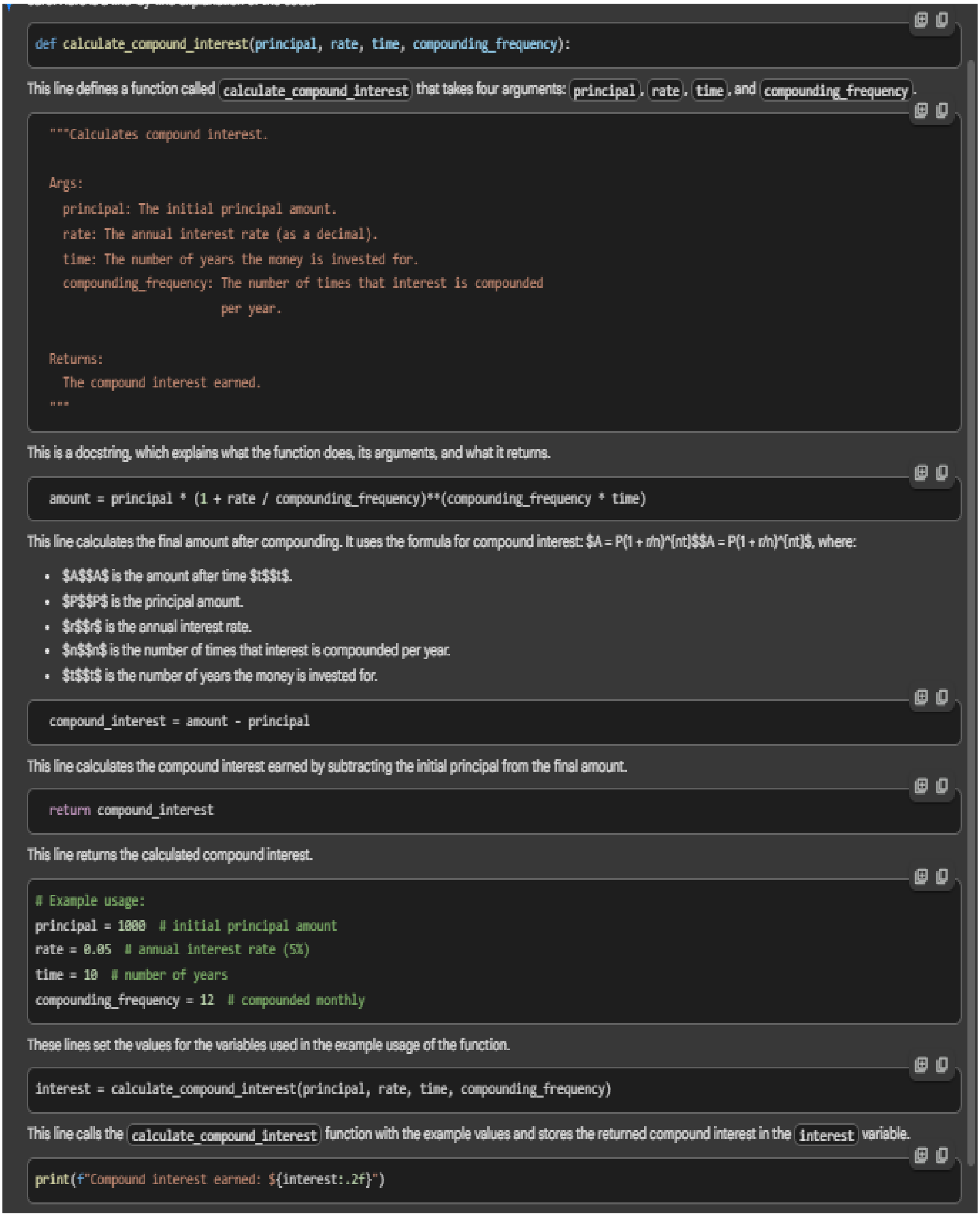
TASK 1:



```
def calculate_compound_interest(principal, rate, time, compounding_frequency):
  """Calculates compound interest.

  Args:
    principal: The initial principal amount.
    rate: The annual interest rate (as a decimal).
    time: The number of years the money is invested for.
    compounding_frequency: The number of times that interest is compounded
                           per year.

  Returns:
    The compound interest earned.
  """
  amount = principal * (1 + rate / compounding_frequency)**(compounding_frequency * time)
  compound_interest = amount - principal
  return compound_interest

# Example usage:
principal = 1000  # initial principal amount
rate = 0.05  # annual interest rate (5%)
time = 10  # number of years
compounding_frequency = 12  # compounded monthly

interest = calculate_compound_interest(principal, rate, time, compounding_frequency)
print(f"Compound interest earned: ${interest:.2f}")
```

```
Compound interest earned: $647.01
```

EXPLANATION :



```
def calculate_compound_interest(principal, rate, time, compounding_frequency):
```

This line defines a function called `calculate_compound_interest` that takes four arguments: `principal`, `rate`, `time`, and `compounding_frequency`.

```
  """Calculates compound interest.

  Args:
    principal: The initial principal amount.
    rate: The annual interest rate (as a decimal).
    time: The number of years the money is invested for.
    compounding_frequency: The number of times that interest is compounded
                         per year.

  Returns:
    The compound interest earned.
  """
```

This is a docstring, which explains what the function does, its arguments, and what it returns.

```
  amount = principal * (1 + rate / compounding_frequency)**(compounding_frequency * time)
```

This line calculates the final amount after compounding. It uses the formula for compound interest: $A = P(1 + r/n)^(nt)$$A = P(1 + r/n)^(nt)$, where:

- $A$$A$ is the amount after time $t$$t$.
- $P$$P$ is the principal amount.
- $r$$r$ is the annual interest rate.
- $n$$n$ is the number of times that interest is compounded per year.
- $t$$t$ is the number of years the money is invested for.

```
  compound_interest = amount - principal
```

This line calculates the compound interest earned by subtracting the initial principal from the final amount.

```
  return compound_interest
```

This line returns the calculated compound interest.

```
# Example usage:
principal = 1000  # initial principal amount
rate = 0.05  # annual interest rate (5%)
time = 10  # number of years
compounding_frequency = 12  # compounded monthly
```

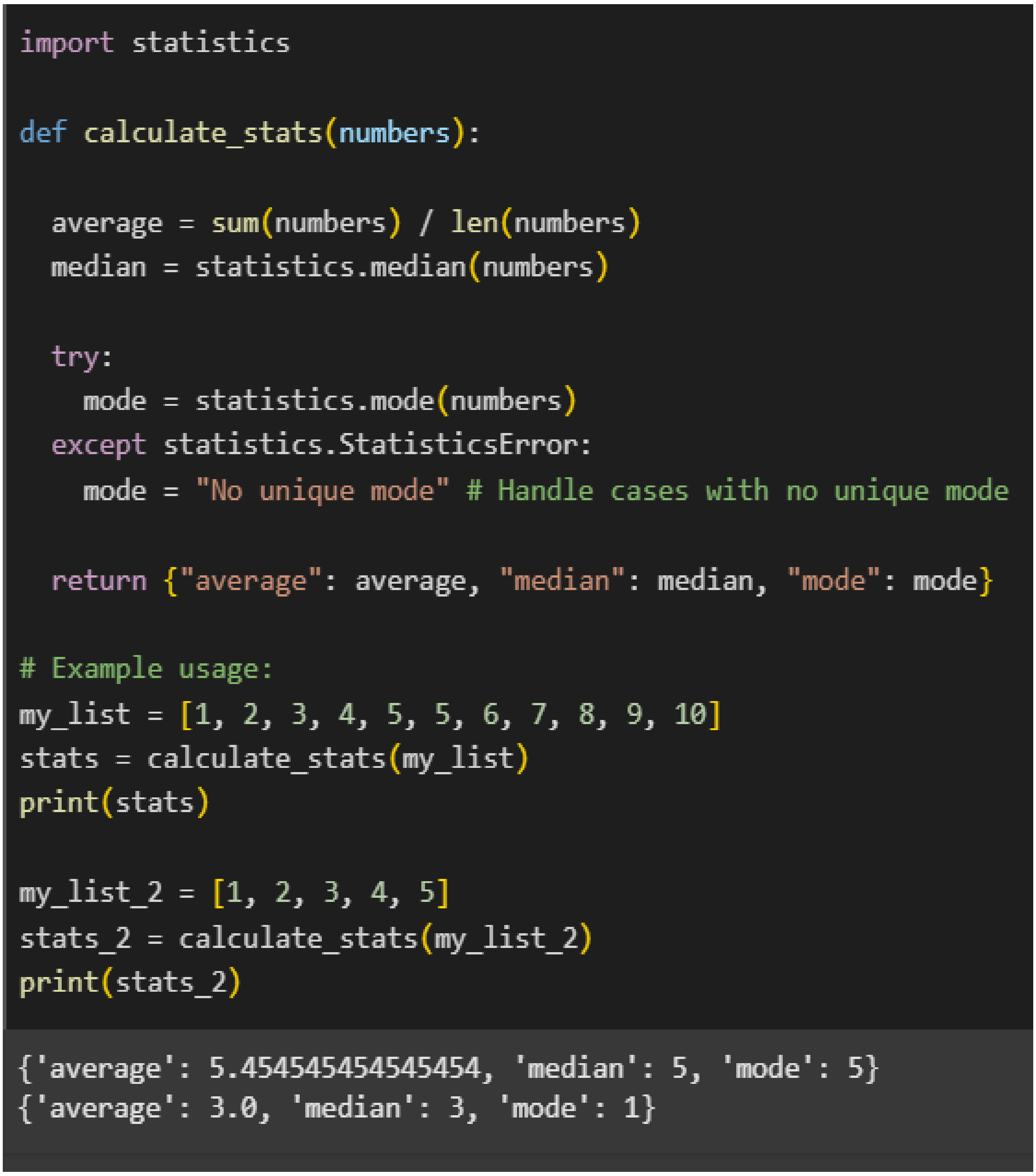
These lines set the values for the variables used in the example usage of the function.

```
interest = calculate_compound_interest(principal, rate, time, compounding_frequency)
```

This line calls the `calculate_compound_interest` function with the example values and stores the returned compound interest in the `interest` variable.

```
print(f"Compound interest earned: ${interest:.2f}")
```

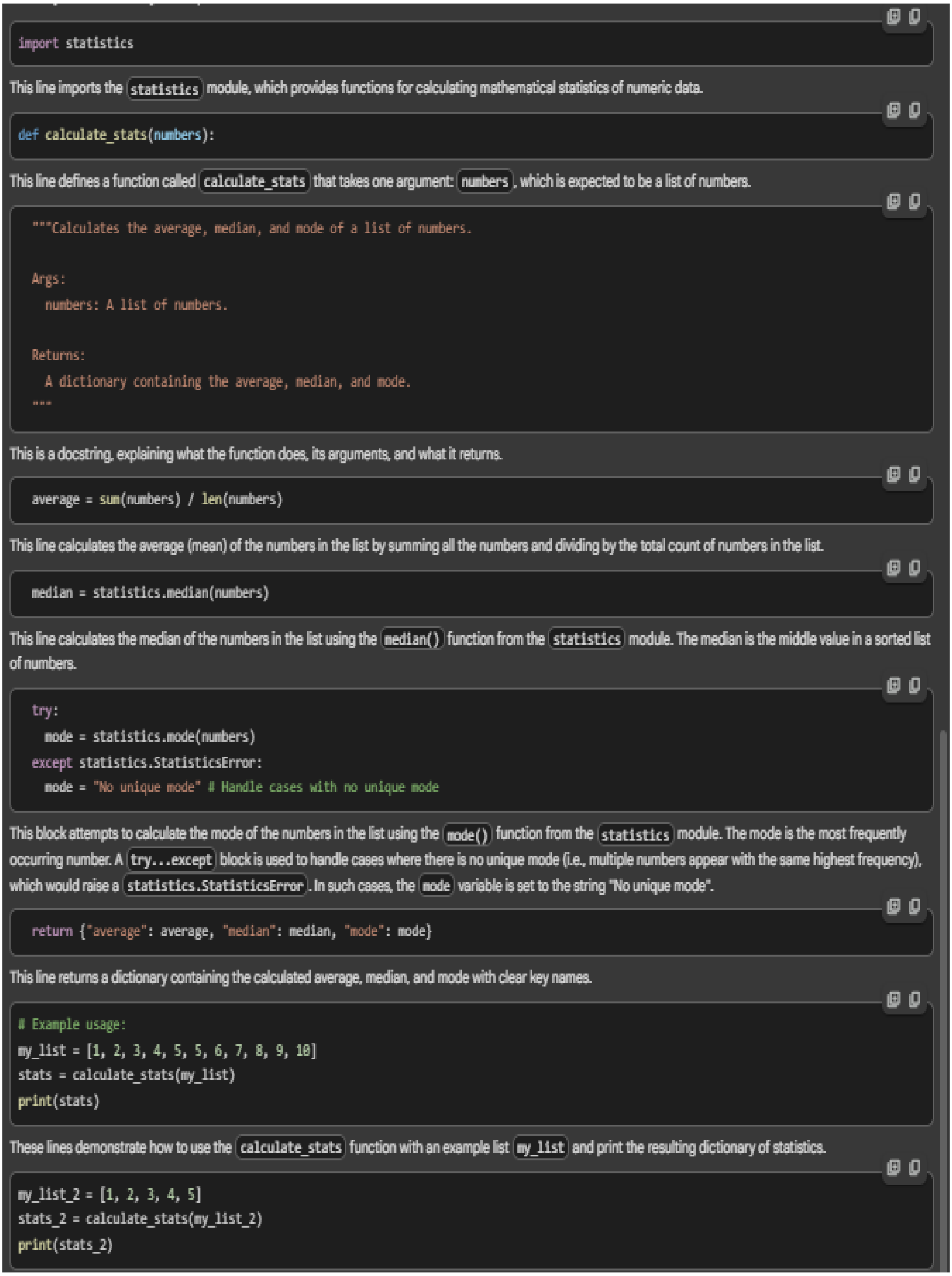
TASK 2:



```
import statistics

def calculate_stats(numbers):

  average = sum(numbers) / len(numbers)
  median = statistics.median(numbers)

  try:
    mode = statistics.mode(numbers)
  except statistics.StatisticsError:
    mode = "No unique mode" # Handle cases with no unique mode

  return {"average": average, "median": median, "mode": mode}

# Example usage:
my_list = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
stats = calculate_stats(my_list)
print(stats)

my_list_2 = [1, 2, 3, 4, 5]
stats_2 = calculate_stats(my_list_2)
print(stats_2)
```

```
{'average': 5.454545454545454, 'median': 5, 'mode': 5}
{'average': 3.0, 'median': 3, 'mode': 1}
```

EXPLANATION:



```
import statistics
```

This line imports the `statistics` module, which provides functions for calculating mathematical statistics of numeric data.

```
def calculate_stats(numbers):
```

This line defines a function called `calculate_stats` that takes one argument: `numbers`, which is expected to be a list of numbers.

```
  """Calculates the average, median, and mode of a list of numbers.

  Args:
    numbers: A list of numbers.

  Returns:
    A dictionary containing the average, median, and mode.
  """
```

This is a docstring, explaining what the function does, its arguments, and what it returns.

```
  average = sum(numbers) / len(numbers)
```

This line calculates the average (mean) of the numbers in the list by summing all the numbers and dividing by the total count of numbers in the list.

```
  median = statistics.median(numbers)
```

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

```
  try:
    mode = statistics.mode(numbers)
  except statistics.StatisticsError:
    mode = "No unique mode" # Handle cases with no unique mode
```

This block attempts to calculate the mode of the numbers in the list using the `mode()` function from the `statistics` module. The mode is the most frequently occurring number. A `try...except` block is used to handle cases where there is no unique mode (i.e., multiple numbers appear with the same highest frequency), which would raise a `statistics.StatisticsError`. In such cases, the `mode` variable is set to the string "No unique mode".

```
  return {"average": average, "median": median, "mode": mode}
```

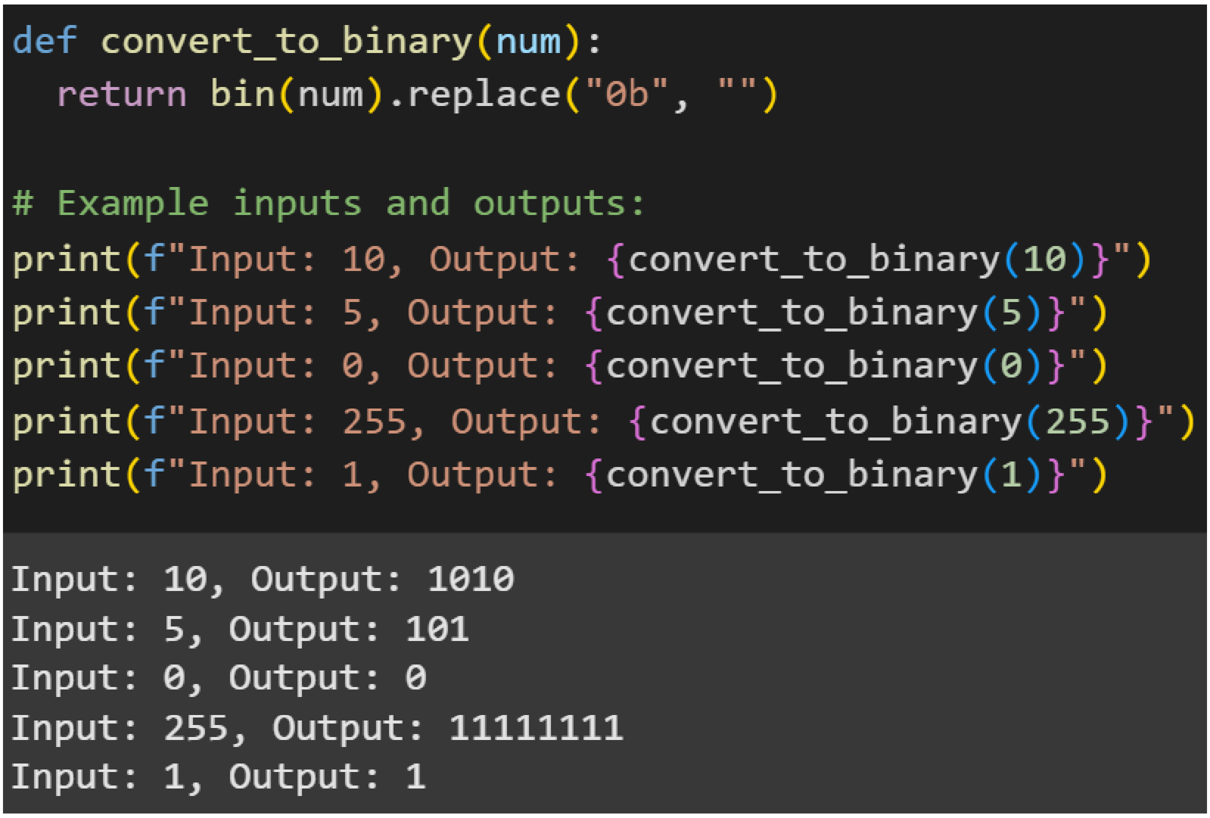
This line returns a dictionary containing the calculated average, median, and mode with clear key names.

```
# Example usage:
my_list = [1, 2, 3, 4, 5, 5, 6, 7, 8, 9, 10]
stats = calculate_stats(my_list)
print(stats)
```

These lines demonstrate how to use the `calculate_stats` function with an example list `my_list` and print the resulting dictionary of statistics.

```
my_list_2 = [1, 2, 3, 4, 5]
stats_2 = calculate_stats(my_list_2)
print(stats_2)
```

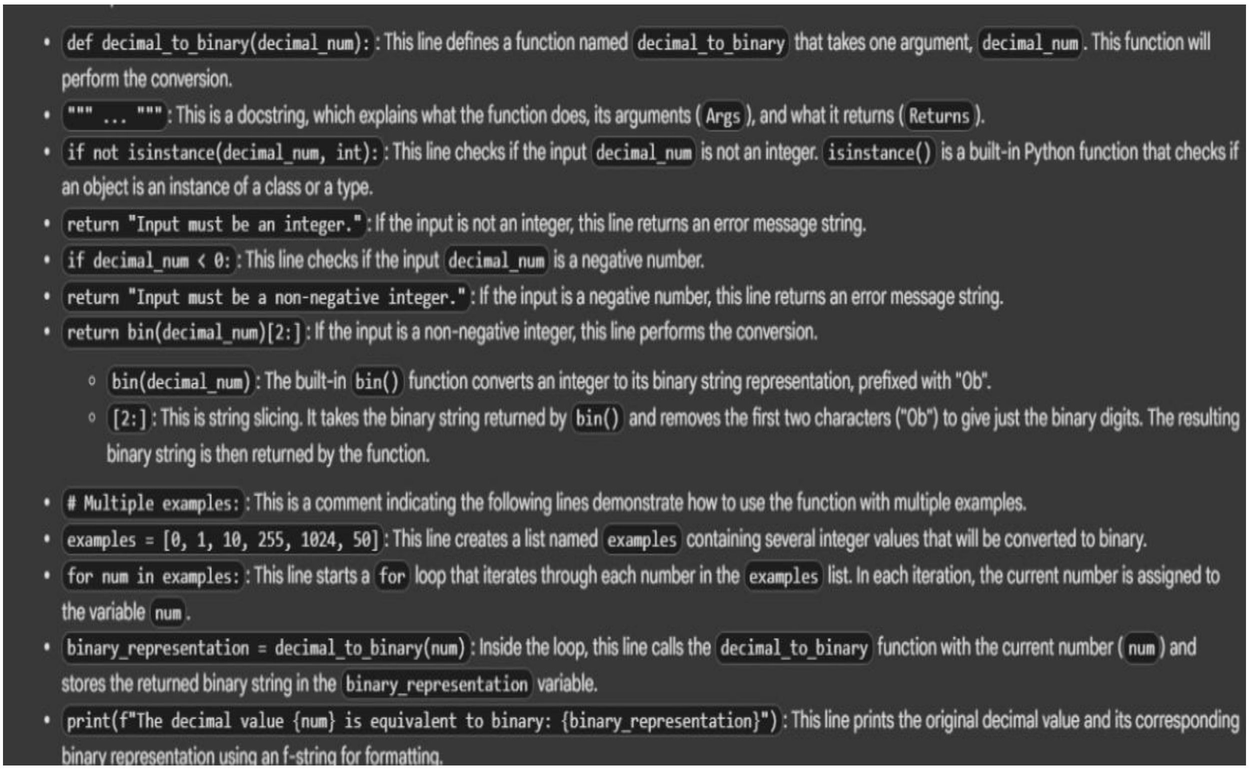
TASK 3 :



```python
def convert_to_binary(num):
  return bin(num).replace("0b", "")

# Example inputs and outputs:
print(f"Input: 10, Output: {convert_to_binary(10)}")
print(f"Input: 5, Output: {convert_to_binary(5)}")
print(f"Input: 0, Output: {convert_to_binary(0)}")
print(f"Input: 255, Output: {convert_to_binary(255)}")
print(f"Input: 1, Output: {convert_to_binary(1)}")
```

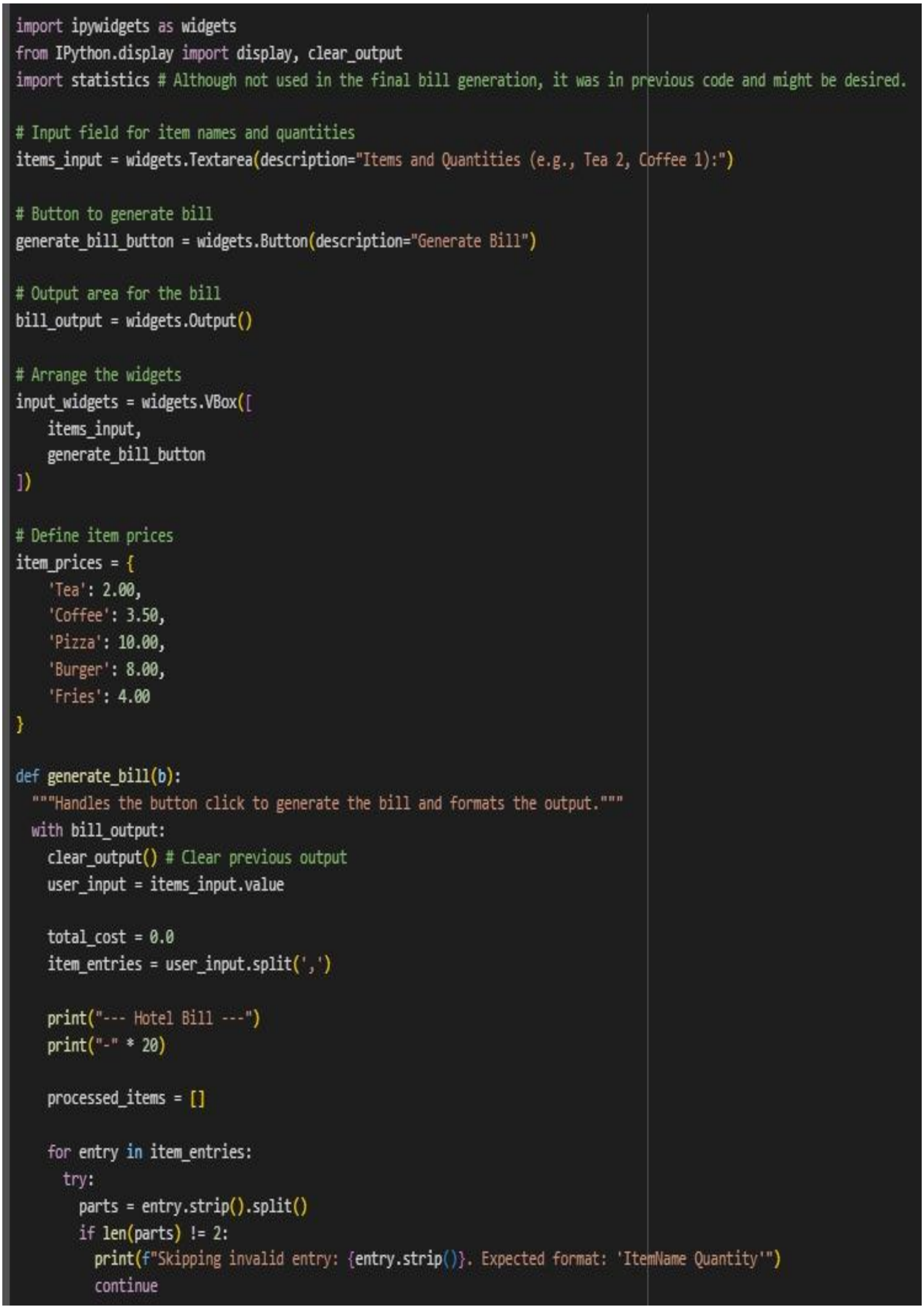
```
Input: 10, Output: 1010
Input: 5, Output: 101
Input: 0, Output: 0
Input: 255, Output: 11111111
Input: 1, Output: 1
```

EXPLANATION :

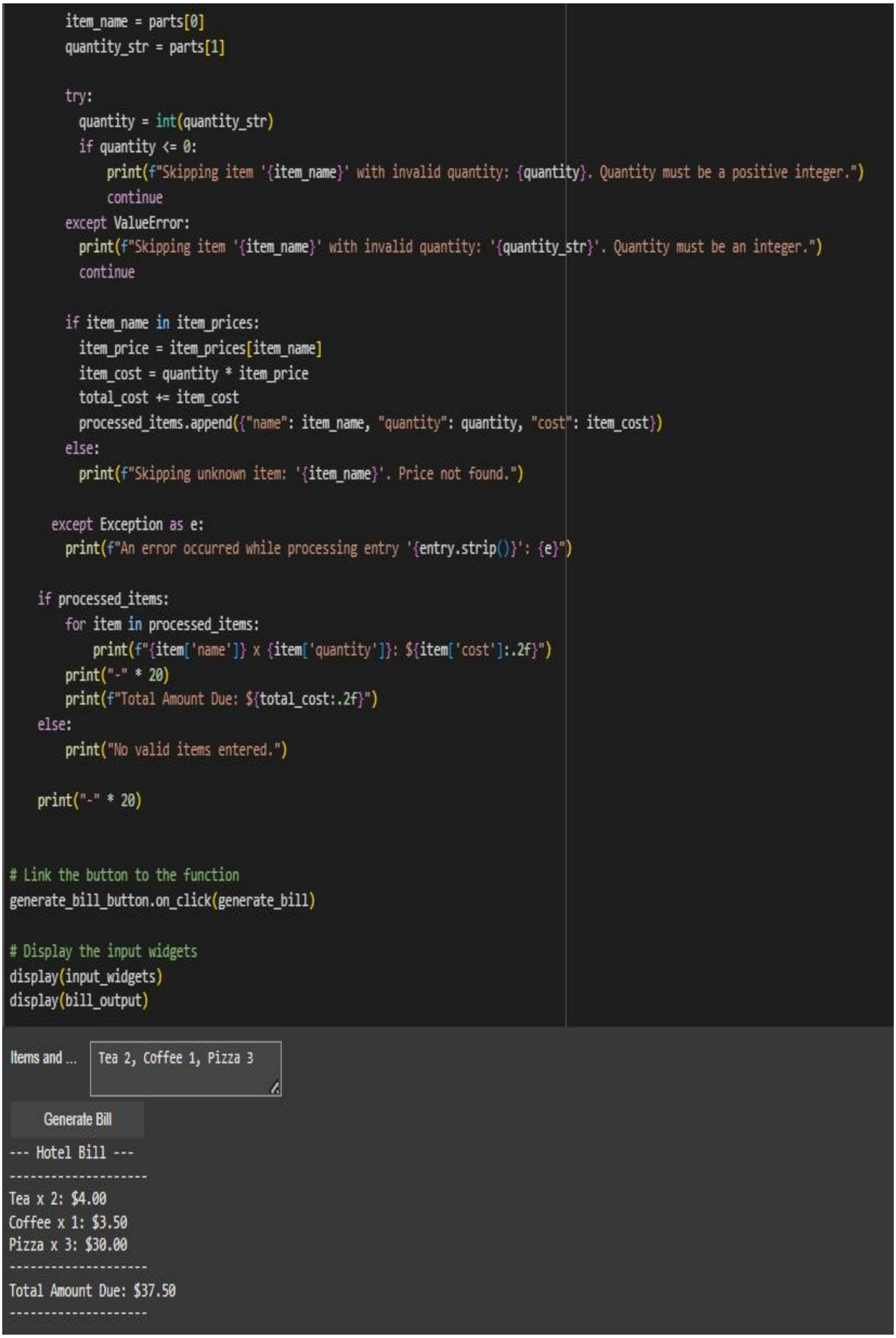


- `def decimal_to_binary(decimal_num):` : This line defines a function named `decimal_to_binary` that takes one argument, `decimal_num`. This function will perform the conversion.
- `""" ... """` : This is a docstring, which explains what the function does, its arguments (`Args`), and what it returns (`Returns`).
- `if not isinstance(decimal_num, int):` : This line checks if the input `decimal_num` is not an integer. `isinstance()` is a built-in Python function that checks if an object is an instance of a class or a type.
- `return "Input must be an integer."` : If the input is not an integer, this line returns an error message string.
- `if decimal_num < 0:` : This line checks if the input `decimal_num` is a negative number.
- `return "Input must be a non-negative integer."` : If the input is a negative number, this line returns an error message string.
- `return bin(decimal_num)[2:]` : If the input is a non-negative integer, this line performs the conversion.
  - `bin(decimal_num)` : The built-in `bin()` function converts an integer to its binary string representation, prefixed with "0b".
  - `[2:]` : This is string slicing. It takes the binary string returned by `bin()` and removes the first two characters ("0b") to give just the binary digits. The resulting binary string is then returned by the function.
- `# Multiple examples:` : This is a comment indicating the following lines demonstrate how to use the function with multiple examples.
- `examples = [0, 1, 10, 255, 1024, 50]` : This line creates a list named `examples` containing several integer values that will be converted to binary.
- `for num in examples:` : This line starts a `for` loop that iterates through each number in the `examples` list. In each iteration, the current number is assigned to the variable `num`.
- `binary_representation = decimal_to_binary(num)` : Inside the loop, this line calls the `decimal_to_binary` function with the current number (`num`) and stores the returned binary string in the `binary_representation` variable.
- `print(f"The decimal value {num} is equivalent to binary: {binary_representation}")` : This line prints the original decimal value and its corresponding binary representation using an f-string for formatting.

TASK 4 :



```
import ipywidgets as widgets
from IPython.display import display, clear_output
import statistics # Although not used in the final bill generation, it was in previous code and might be desired.

# Input field for item names and quantities
items_input = widgets.Textarea(description="Items and Quantities (e.g., Tea 2, Coffee 1):")

# Button to generate bill
generate_bill_button = widgets.Button(description="Generate Bill")

# Output area for the bill
bill_output = widgets.Output()

# Arrange the widgets
input_widgets = widgets.VBox([
    items_input,
    generate_bill_button
])

# Define item prices
item_prices = {
    'Tea': 2.00,
    'Coffee': 3.50,
    'Pizza': 10.00,
    'Burger': 8.00,
    'Fries': 4.00
}

def generate_bill(b):
  """Handles the button click to generate the bill and formats the output."""
  with bill_output:
    clear_output() # Clear previous output
    user_input = items_input.value

    total_cost = 0.0
    item_entries = user_input.split(',')

    print("--- Hotel Bill ---")
    print("-" * 20)

    processed_items = []

    for entry in item_entries:
      try:
        parts = entry.strip().split()
        if len(parts) != 2:
          print(f"Skipping invalid entry: {entry.strip()}. Expected format: 'ItemName Quantity'")
          continue
```

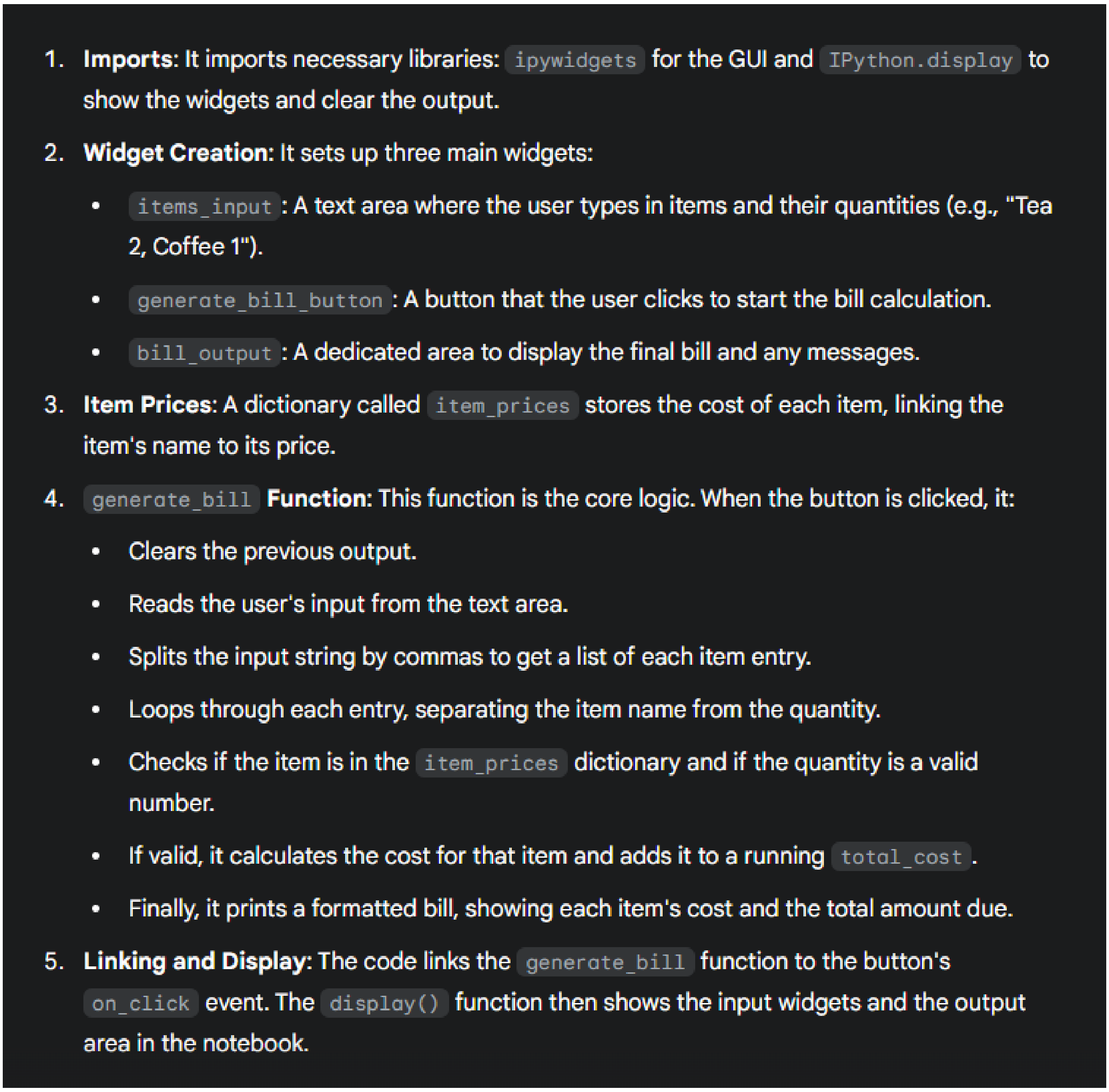


```
        item_name = parts[0]
        quantity_str = parts[1]

        try:
          quantity = int(quantity_str)
          if quantity <= 0:
              print(f"Skipping item '{item_name}' with invalid quantity: {quantity}. Quantity must be a positive integer.")
              continue
        except ValueError:
          print(f"Skipping item '{item_name}' with invalid quantity: '{quantity_str}'. Quantity must be an integer.")
          continue

        if item_name in item_prices:
          item_price = item_prices[item_name]
          item_cost = quantity * item_price
          total_cost += item_cost
          processed_items.append({"name": item_name, "quantity": quantity, "cost": item_cost})
        else:
          print(f"Skipping unknown item: '{item_name}'. Price not found.")

      except Exception as e:
        print(f"An error occurred while processing entry '{entry.strip()}': {e}")

    if processed_items:
        for item in processed_items:
            print(f"{item['name']} x {item['quantity']}: ${item['cost']:.2f}")
        print("-" * 20)
        print(f"Total Amount Due: ${total_cost:.2f}")
    else:
        print("No valid items entered.")

    print("-" * 20)


# Link the button to the function
generate_bill_button.on_click(generate_bill)

# Display the input widgets
display(input_widgets)
display(bill_output)
```

Items and ... Tea 2, Coffee 1, Pizza 3

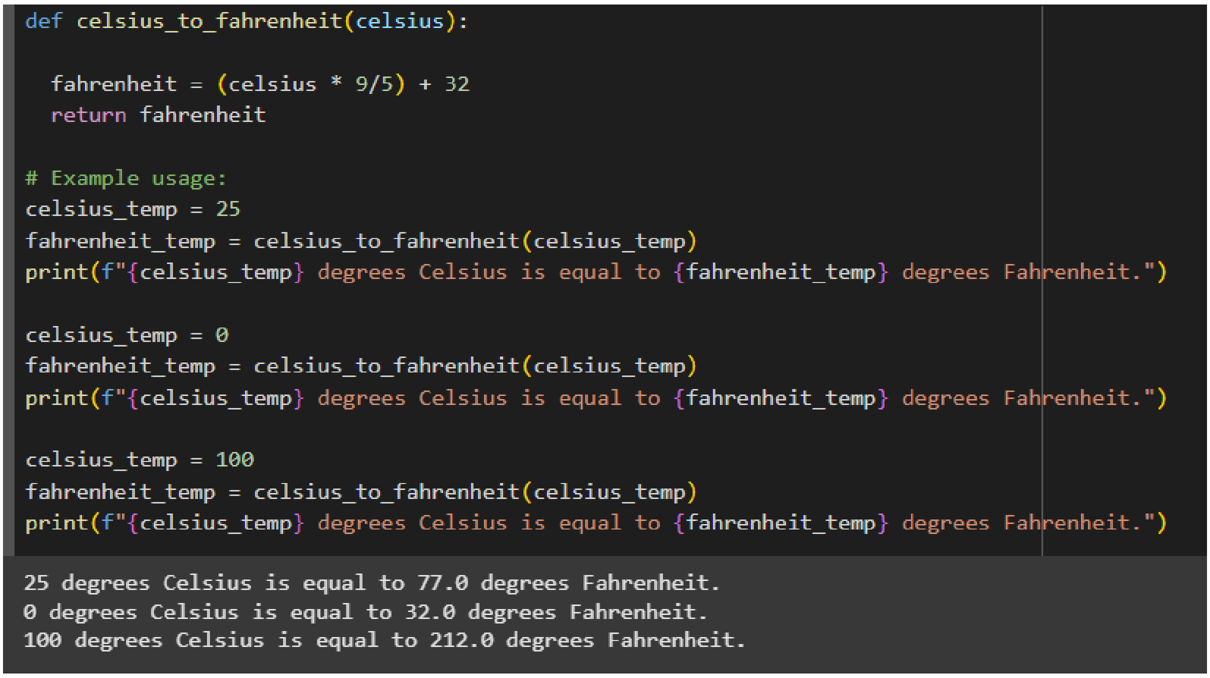
```
--- Hotel Bill ---
--------------------
Tea x 2: $4.00
Coffee x 1: $3.50
Pizza x 3: $30.00
--------------------
Total Amount Due: $37.50
--------------------
```

EXPLANATION :



1. **Imports**: It imports necessary libraries: `ipywidgets` for the GUI and `IPython.display` to show the widgets and clear the output.
2. **Widget Creation:** It sets up three main widgets:
   - `items_input` : A text area where the user types in items and their quantities (e.g., "Tea 2, Coffee 1").
   - `generate_bill_button` : A button that the user clicks to start the bill calculation.
   - `bill_output` : A dedicated area to display the final bill and any messages.
3. **Item Prices**: A dictionary called `item_prices` stores the cost of each item, linking the item's name to its price.
4. `generate_bill` **Function:** This function is the core logic. When the button is clicked, it:
   - Clears the previous output.
   - Reads the user's input from the text area.
   - Splits the input string by commas to get a list of each item entry.
   - Loops through each entry, separating the item name from the quantity.
   - Checks if the item is in the `item_prices` dictionary and if the quantity is a valid number.
   - If valid, it calculates the cost for that item and adds it to a running `total_cost` .
   - Finally, it prints a formatted bill, showing each item's cost and the total amount due.
5. **Linking and Display:** The code links the `generate_bill` function to the button's `on_click` event. The `display()` function then shows the input widgets and the output area in the notebook.

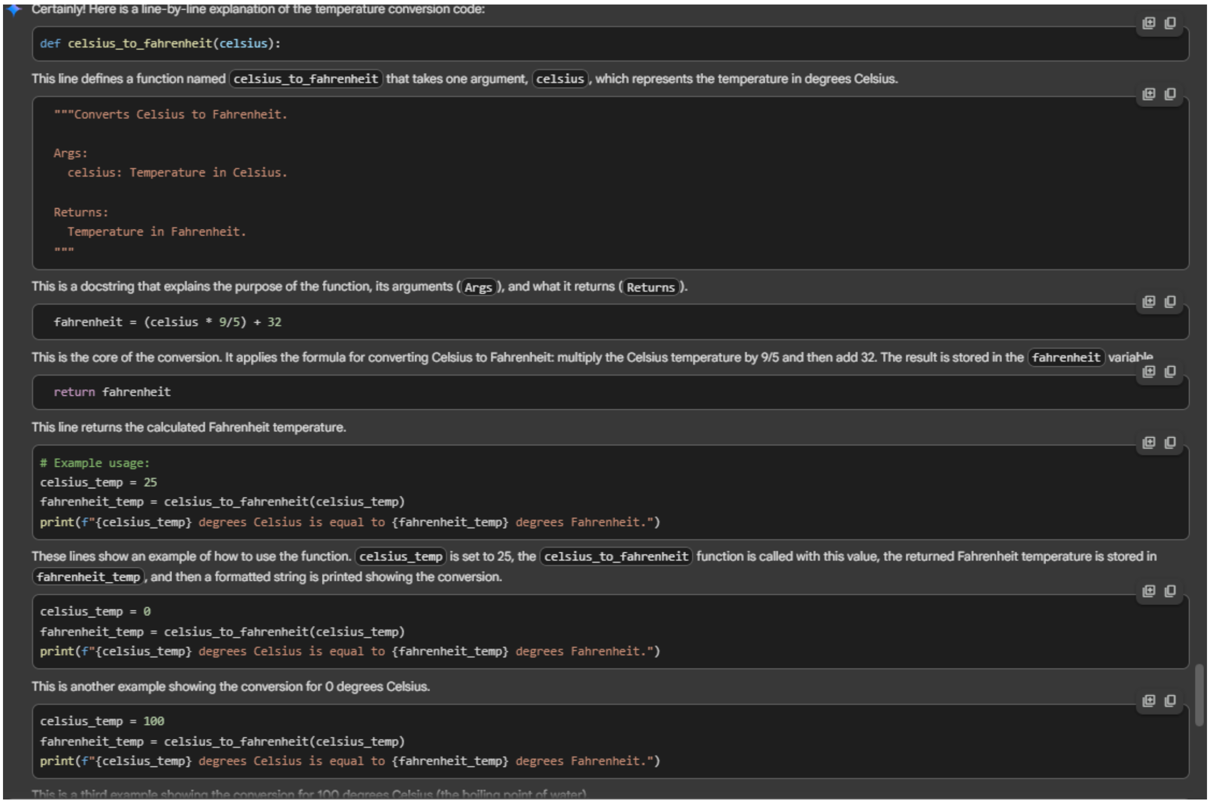
TASK 5 :



```
def celsius_to_fahrenheit(celsius):

  fahrenheit = (celsius * 9/5) + 32
  return fahrenheit

# Example usage:
celsius_temp = 25
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")

celsius_temp = 0
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")

celsius_temp = 100
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")
```

```
25 degrees Celsius is equal to 77.0 degrees Fahrenheit.
0 degrees Celsius is equal to 32.0 degrees Fahrenheit.
100 degrees Celsius is equal to 212.0 degrees Fahrenheit.
```

EXPLANATION :



Certainly! Here is a line-by-line explanation of the temperature conversion code:

```
def celsius_to_fahrenheit(celsius):
```

This line defines a function named `celsius_to_fahrenheit` that takes one argument, `celsius`, which represents the temperature in degrees Celsius.

```
  """Converts Celsius to Fahrenheit.

  Args:
    celsius: Temperature in Celsius.

  Returns:
    Temperature in Fahrenheit.
  """
```

This is a docstring that explains the purpose of the function, its arguments (`Args`), and what it returns (`Returns`).

```
  fahrenheit = (celsius * 9/5) + 32
```

This is the core of the conversion. It applies the formula for converting Celsius to Fahrenheit: multiply the Celsius temperature by 9/5 and then add 32. The result is stored in the `fahrenheit` variable

```
  return fahrenheit
```

This line returns the calculated Fahrenheit temperature.

```
# Example usage:
celsius_temp = 25
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")
```

These lines show an example of how to use the function. `celsius_temp` is set to 25, the `celsius_to_fahrenheit` function is called with this value, the returned Fahrenheit temperature is stored in `fahrenheit_temp`, and then a formatted string is printed showing the conversion.

```
celsius_temp = 0
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")
```

This is another example showing the conversion for 0 degrees Celsius.

```
celsius_temp = 100
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp} degrees Celsius is equal to {fahrenheit_temp} degrees Fahrenheit.")
```

This is a third example showing the conversion for 100 degrees Celsius (the boiling point of water)