

LAB NAME : AI ASSISTED CODING

LAB NUMBER :02

ROLL NO :2503A51L16

BRANCH : CSE

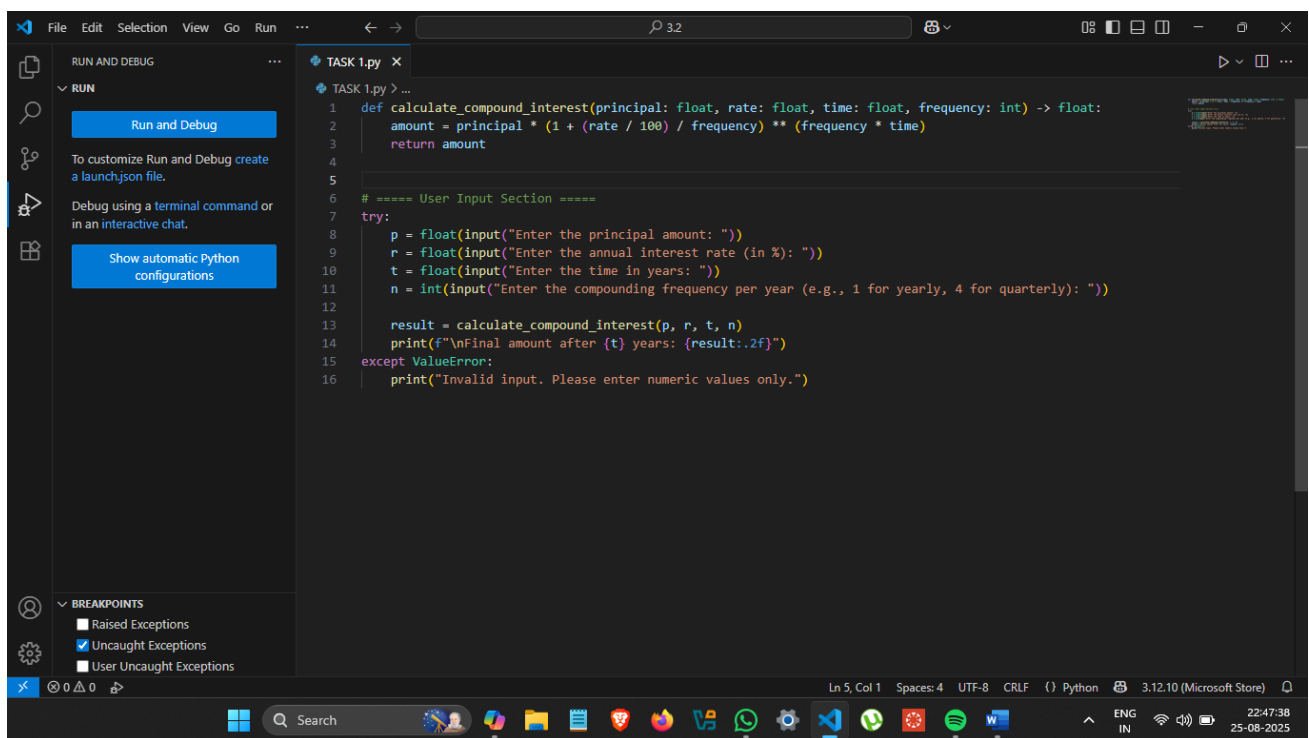
NAME : K.JASHUVA

TASK 1

Task Description: Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example

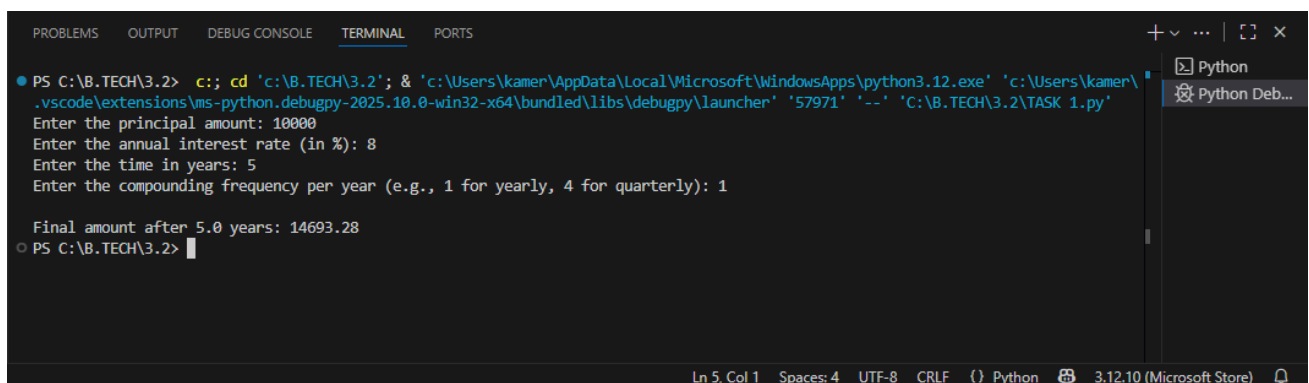
PROMPT: Generate a Python function named `calculate_compound_interest` that takes principal, annual interest rate, time in years, and compounding frequency as parameters.

CODE:



```
1 def calculate_compound_interest(principal: float, rate: float, time: float, frequency: int) -> float:
2     amount = principal * (1 + (rate / 100) / frequency) ** (frequency * time)
3     return amount
4
5
6 # ===== User Input Section =====
7 try:
8     p = float(input("Enter the principal amount: "))
9     r = float(input("Enter the annual interest rate (in %): "))
10    t = float(input("Enter the time in years: "))
11    n = int(input("Enter the compounding frequency per year (e.g., 1 for yearly, 4 for quarterly): "))
12
13    result = calculate_compound_interest(p, r, t, n)
14    print(f"\nFinal amount after {t} years: {result:.2f}")
15 except ValueError:
16    print("Invalid input. Please enter numeric values only.")
```

OUTPUT:



```
PS C:\B.TECH\3.2> c:: cd 'c:\B.TECH\3.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\
.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '57971' '--' 'C:\B.TECH\3.2\TASK 1.py'
Enter the principal amount: 10000
Enter the annual interest rate (in %): 8
Enter the time in years: 5
Enter the compounding frequency per year (e.g., 1 for yearly, 4 for quarterly): 1

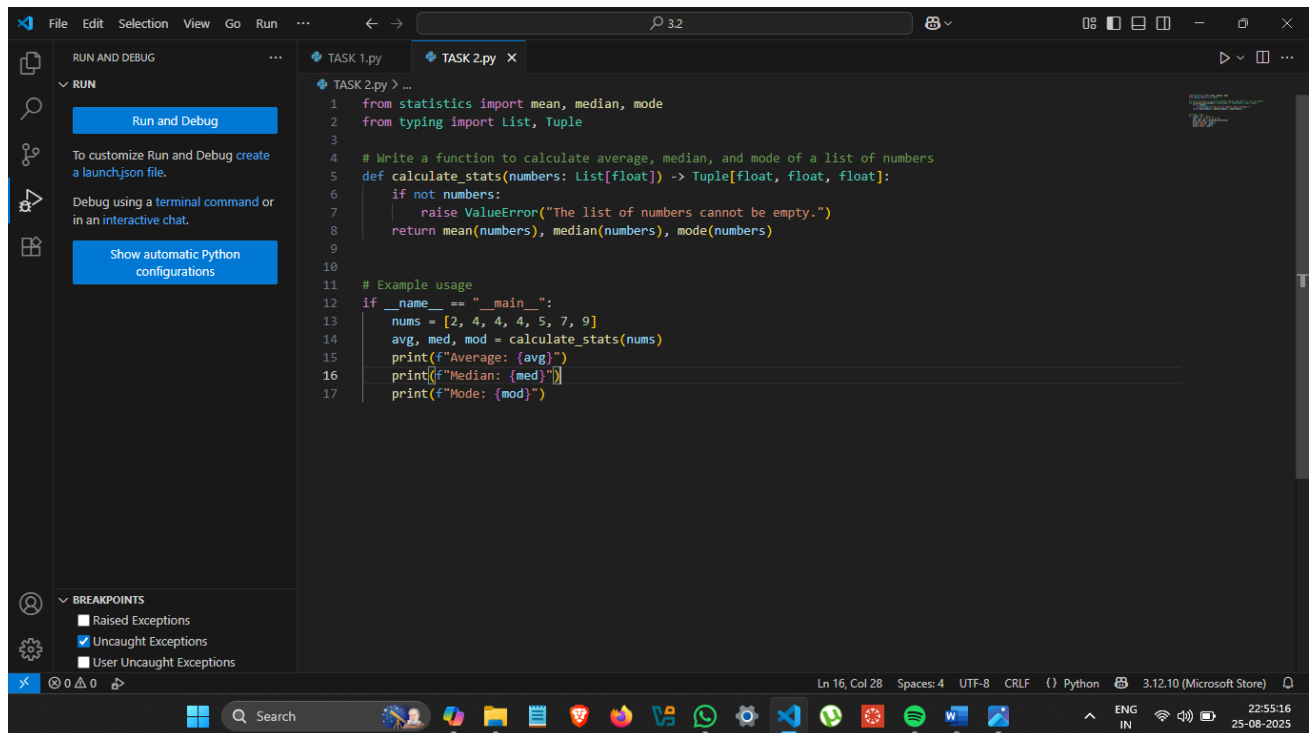
Final amount after 5.0 years: 14693.28
PS C:\B.TECH\3.2>
```

TASK 2

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

PROMPT: Generate a Python function with type hints that calculates the average, median, and mode of a list of numbers, that includes a clear docstring, and handles empty lists or invalid inputs gracefully.

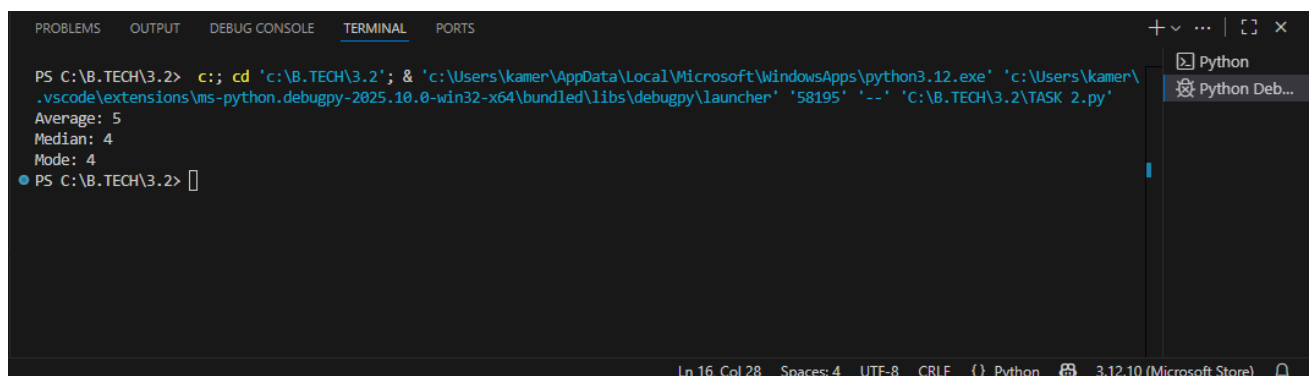
CODE:



The screenshot shows the Visual Studio Code editor with a Python file named 'TASK 2.py'. The code defines a function 'calculate_stats' that takes a list of numbers and returns a tuple of average, median, and mode. It includes a docstring and a ValueError for empty lists. The main block uses the function on a sample list [2, 4, 4, 4, 5, 7, 9] and prints the results.

```
1 from statistics import mean, median, mode
2 from typing import List, Tuple
3
4 # Write a function to calculate average, median, and mode of a list of numbers
5 def calculate_stats(numbers: List[float]) -> Tuple[float, float, float]:
6     if not numbers:
7         raise ValueError("The list of numbers cannot be empty.")
8     return mean(numbers), median(numbers), mode(numbers)
9
10
11 # Example usage
12 if __name__ == "__main__":
13     nums = [2, 4, 4, 4, 5, 7, 9]
14     avg, med, mod = calculate_stats(nums)
15     print(f"Average: {avg}")
16     print(f"Median: {med}")
17     print(f"Mode: {mod}")
```

OUTPUT:



The screenshot shows the VS Code terminal with the output of the Python script. The output displays the average, median, and mode of the list [2, 4, 4, 4, 5, 7, 9].

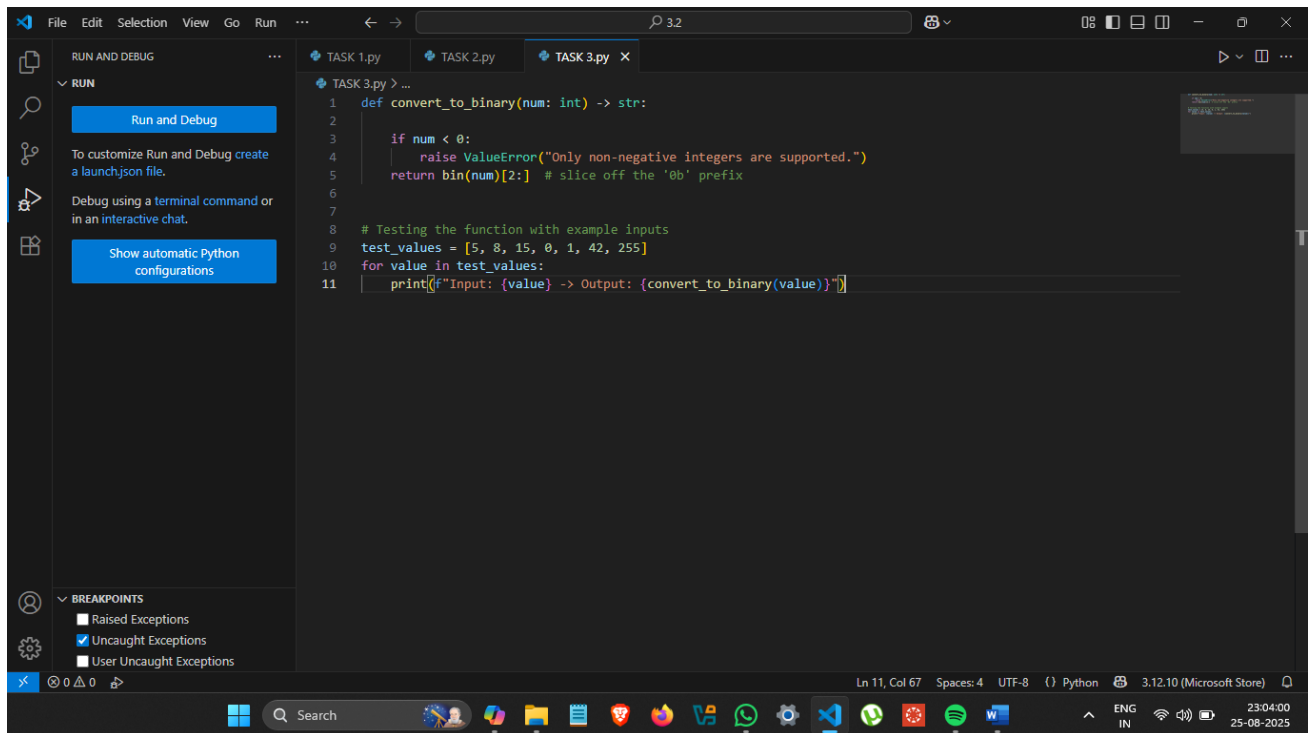
```
PS C:\B.TECH\3.2> c:\; cd 'c:\B.TECH\3.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\
.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58195' '--' 'c:\B.TECH\3.2\TASK 2.py'
Average: 5
Median: 4
Mode: 4
PS C:\B.TECH\3.2> 
```

TASK 3

Task Description: Provide multiple examples of input-output to the AI for `convert_to_binary(num)` function. Observe how AI uses few-shot prompting to generalize.

PROMPT: Provide multiple examples of input-output for the `convert_to_binary(num)` function to help the AI generalize using few-shot prompting.

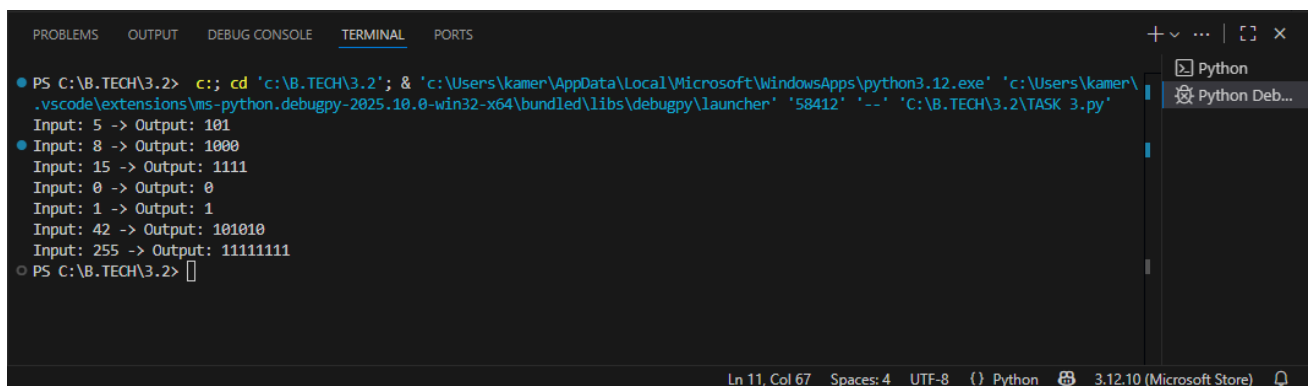
CODE:



The screenshot shows the Visual Studio Code editor with a Python file named `TASK 3.py`. The code defines a function `convert_to_binary(num: int) -> str:`. It includes a check for non-negative integers, raising a `ValueError` if `num < 0`. The function uses `bin(num)[2:]` to convert the number to binary and slice off the `'0b'` prefix. Below the function, there is a test suite with example inputs: `test_values = [5, 8, 15, 0, 1, 42, 255]`. A loop iterates over these values, printing the input and the output of the function: `print(f"Input: {value} -> Output: {convert_to_binary(value)}")`. The left sidebar shows the 'RUN AND DEBUG' panel with options to 'Run and Debug', 'Show automatic Python configurations', and 'Breakpoints'. The bottom status bar indicates the file is at line 11, column 67, using UTF-8 encoding, and the Python interpreter is 3.12.10 (Microsoft Store).

```
1 def convert_to_binary(num: int) -> str:
2
3     if num < 0:
4         raise ValueError("Only non-negative integers are supported.")
5     return bin(num)[2:] # slice off the '0b' prefix
6
7
8 # Testing the function with example inputs
9 test_values = [5, 8, 15, 0, 1, 42, 255]
10 for value in test_values:
11     print(f"Input: {value} -> Output: {convert_to_binary(value)}")
```

OUTPUT:



The screenshot shows the Visual Studio Code terminal window with the output of the Python script. The command prompt is `PS C:\B.TECH\3.2> c:; cd 'c:\B.TECH\3.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58412' '--' 'C:\B.TECH\3.2\TASK 3.py'`. The output shows the function's results for each input value: `Input: 5 -> Output: 101`, `Input: 8 -> Output: 1000`, `Input: 15 -> Output: 1111`, `Input: 0 -> Output: 0`, `Input: 1 -> Output: 1`, `Input: 42 -> Output: 101010`, and `Input: 255 -> Output: 11111111`. The terminal window also shows the 'TERMINAL' tab selected, and the bottom status bar indicates the file is at line 11, column 67, using UTF-8 encoding, and the Python interpreter is 3.12.10 (Microsoft Store).

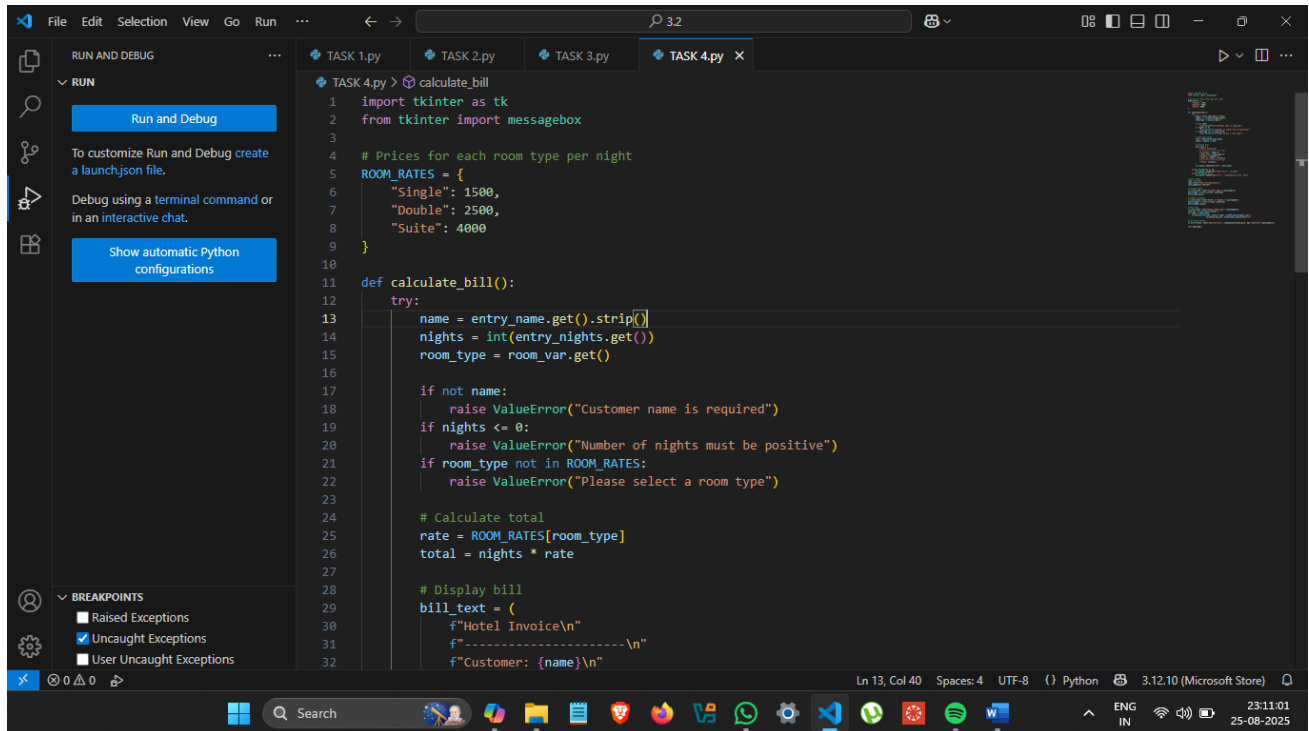
```
PS C:\B.TECH\3.2> c:; cd 'c:\B.TECH\3.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\.vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '58412' '--' 'C:\B.TECH\3.2\TASK 3.py'
Input: 5 -> Output: 101
Input: 8 -> Output: 1000
Input: 15 -> Output: 1111
Input: 0 -> Output: 0
Input: 1 -> Output: 1
Input: 42 -> Output: 101010
Input: 255 -> Output: 11111111
PS C:\B.TECH\3.2>
```

TASK 4

Task Description: Create a user interface for a hotel to generate bill based on customer requirements.

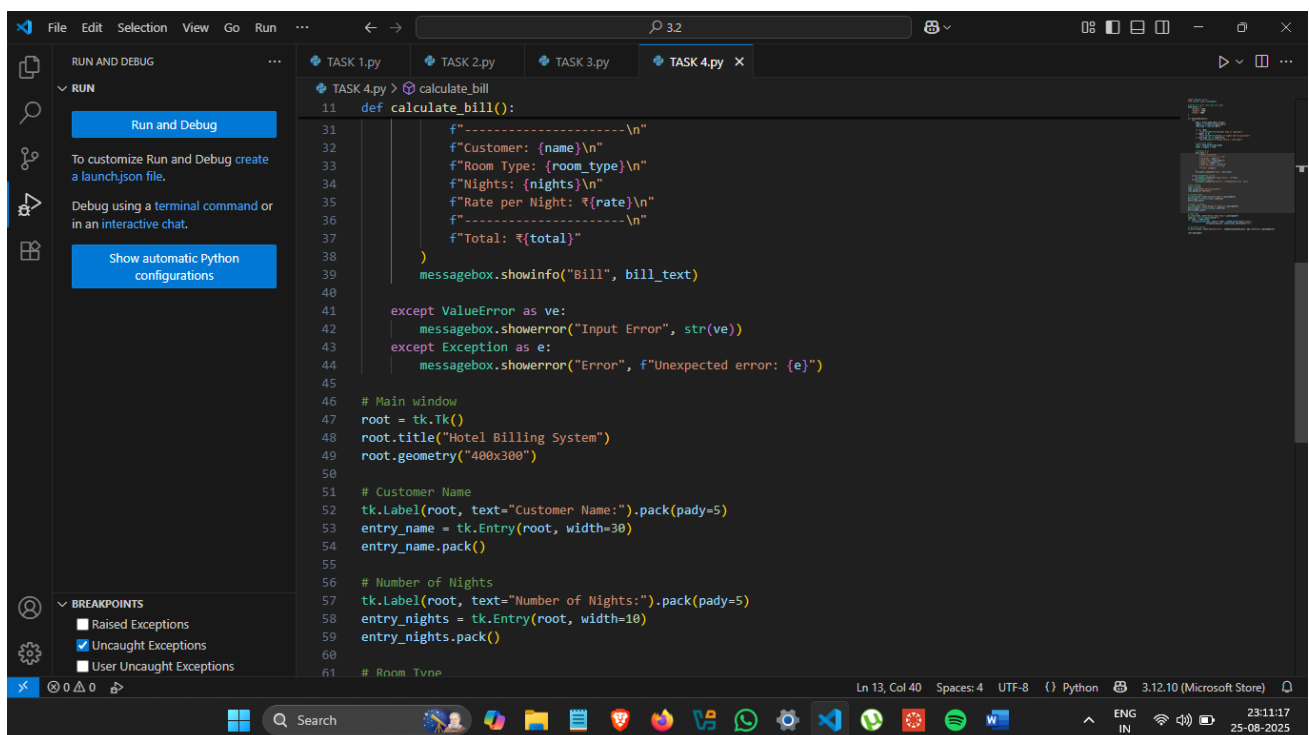
PROMPT: Create a user interface for a hotel billing system that generates a bill based on customer requirements.

CODE:



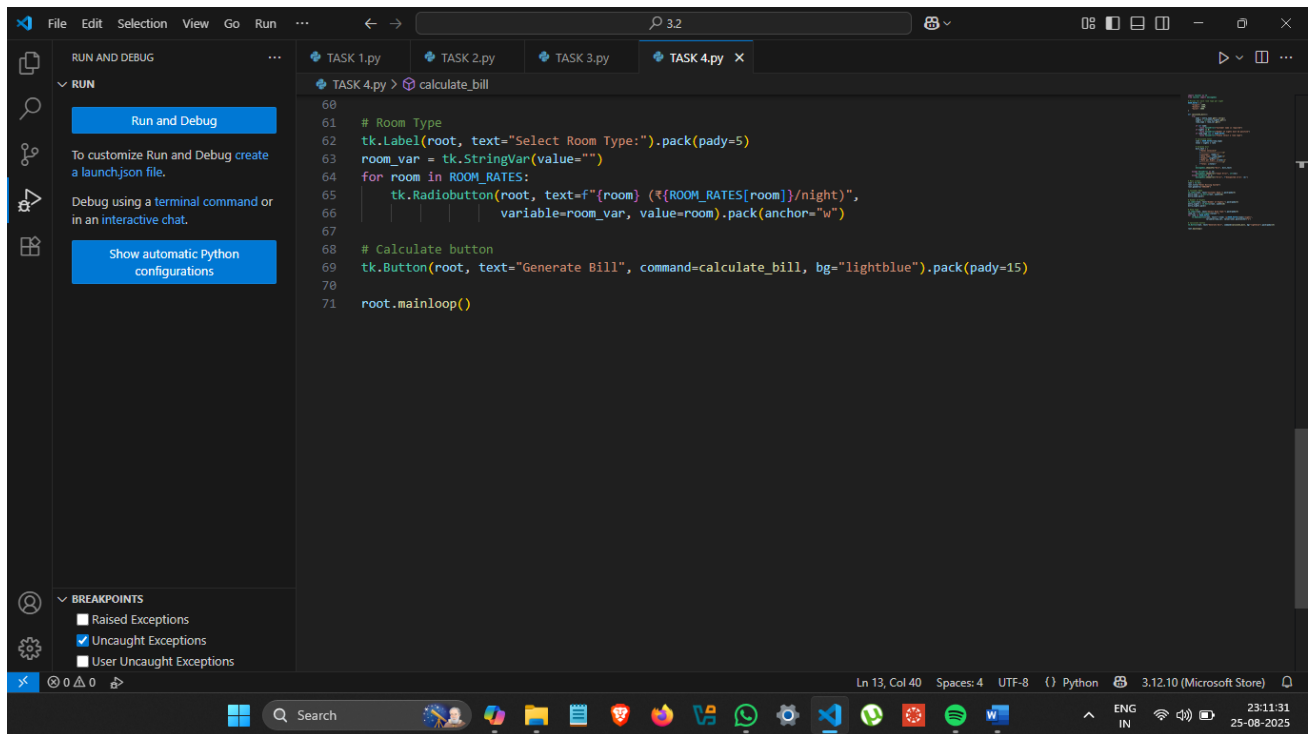
The screenshot shows the Visual Studio Code editor with a Python file named TASK 4.py. The code defines a function `calculate_bill()` that takes user input for name, nights, and room type, calculates the total bill, and displays it. The `ROOM_RATES` dictionary is defined as follows:

```
1 import tkinter as tk
2 from tkinter import messagebox
3
4 # Prices for each room type per night
5 ROOM_RATES = {
6     "Single": 1500,
7     "Double": 2500,
8     "Suite": 4000
9 }
10
11 def calculate_bill():
12     try:
13         name = entry_name.get().strip()
14         nights = int(entry_nights.get())
15         room_type = room_var.get()
16
17         if not name:
18             raise ValueError("Customer name is required")
19         if nights <= 0:
20             raise ValueError("Number of nights must be positive")
21         if room_type not in ROOM_RATES:
22             raise ValueError("Please select a room type")
23
24         # Calculate total
25         rate = ROOM_RATES[room_type]
26         total = nights * rate
27
28         # Display bill
29         bill_text = (
30             f"Hotel Invoice\n"
31             f"-----\n"
32             f"Customer: {name}\n"
```

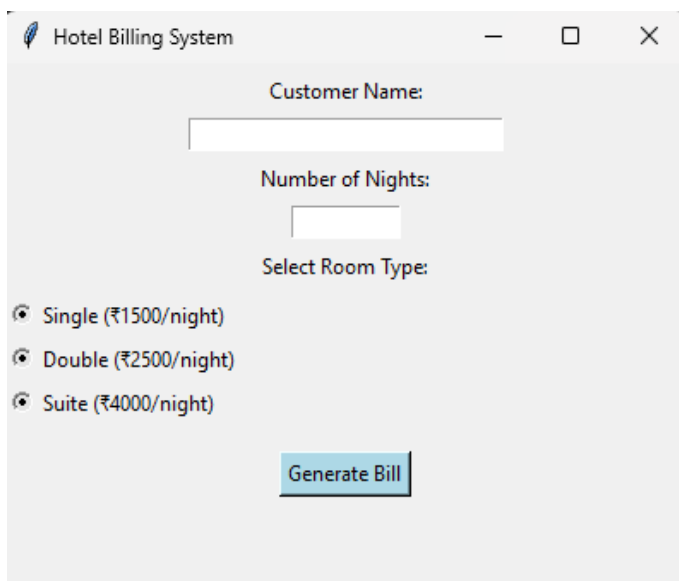


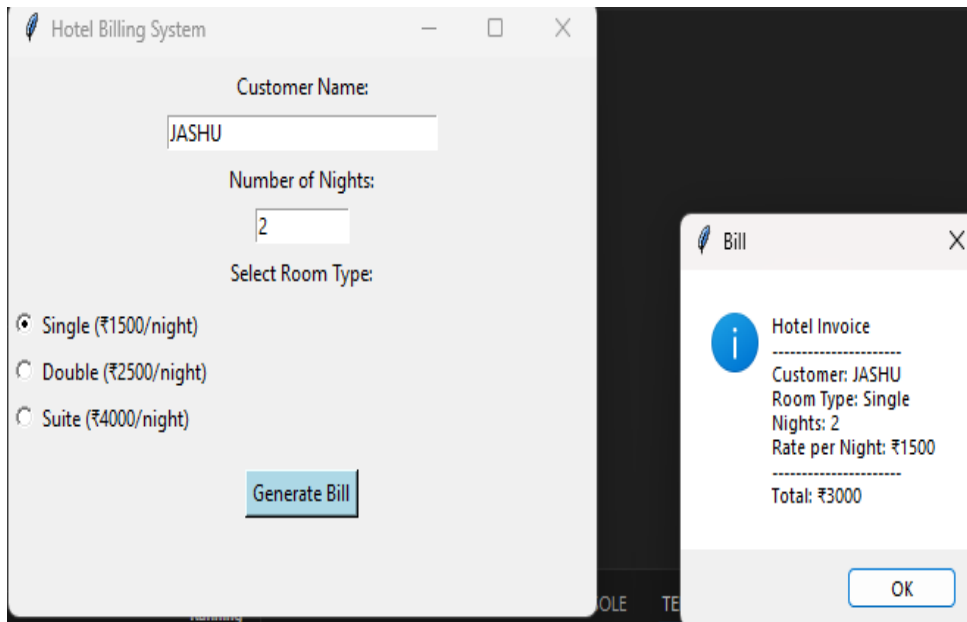
The screenshot shows the Visual Studio Code editor with the same Python file, displaying the continuation of the `calculate_bill()` function and the main window setup. The function continues with error handling and a call to `messagebox.showinfo()`. The main window is created using `tk.Tk()` and titled "Hotel Billing System".

```
33         f"Room Type: {room_type}\n"
34         f"Nights: {nights}\n"
35         f"Rate per Night: ₹{rate}\n"
36         f"-----\n"
37         f"Total: ₹{total}"
38     )
39     messagebox.showinfo("Bill", bill_text)
40
41 except ValueError as ve:
42     messagebox.showerror("Input Error", str(ve))
43 except Exception as e:
44     messagebox.showerror("Error", f"Unexpected error: {e}")
45
46 # Main window
47 root = tk.Tk()
48 root.title("Hotel Billing System")
49 root.geometry("400x300")
50
51 # Customer Name
52 tk.Label(root, text="Customer Name:").pack(pady=5)
53 entry_name = tk.Entry(root, width=30)
54 entry_name.pack()
55
56 # Number of Nights
57 tk.Label(root, text="Number of Nights:").pack(pady=5)
58 entry_nights = tk.Entry(root, width=10)
59 entry_nights.pack()
60
61 # Room Type
```



OUTPUT:





TASK 5

Task Description: Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

PROMPT: Give a Python Program to convert temperatures between Celsius, Fahrenheit, and Kelvin.

CODE:

```

1  def convert_temperature(value: float, from_unit: str, to_unit: str) -> float:
2      from_unit = from_unit.upper()
3      to_unit = to_unit.upper()
4
5      valid_units = {'C', 'F', 'K'}
6      if from_unit not in valid_units or to_unit not in valid_units:
7          raise ValueError("Units must be 'C', 'F', or 'K'.")
8
9      # Convert input to Celsius first
10     if from_unit == 'C':
11         celsius = value
12     elif from_unit == 'F':
13         celsius = (value - 32) * 5 / 9
14     elif from_unit == 'K':
15         celsius = value - 273.15
16
17     # Convert Celsius to target unit
18     if to_unit == 'C':
19         return celsius
20     elif to_unit == 'F':
21         return (celsius * 9 / 5) + 32
22     elif to_unit == 'K':
23         return celsius + 273.15
24
25
26     # ===== Example Usage =====
27     if __name__ == "__main__":
28         try:
29             temp_value = float(input("Enter the temperature value: "))
30             from_u = input("Enter the unit to convert from (C/F/K): ").strip()
31             to_u = input("Enter the unit to convert to (C/F/K): ").strip()
32

```

The screenshot shows the Visual Studio Code editor interface. The left sidebar contains the 'RUN AND DEBUG' panel with options to 'Run and Debug', 'To customize Run and Debug create a launch.json file.', 'Debug using a terminal command or in an interactive chat.', and 'Show automatic Python configurations'. The main editor area displays a Python file named 'TASK 5.py' with the following code:

```
24  
25  
26 # ===== Example Usage =====  
27 if __name__ == "__main__":  
28     try:  
29         temp_value = float(input("Enter the temperature value: "))  
30         from_u = input("Enter the unit to convert from (C/F/K): ").strip()  
31         to_u = input("Enter the unit to convert to (C/F/K): ").strip()  
32  
33         result = convert_temperature(temp_value, from_u, to_u)  
34         print(f"{temp_value}°{from_u.upper()} = {result:.2f}°{to_u.upper()}")  
35  
36     except ValueError as e:  
37         print(f"Error: {e}")
```

The bottom status bar indicates 'Ln 37, Col 29', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and '3.12.10 (Microsoft Store)'.

OUTPUT:

The screenshot shows the 'TERMINAL' panel in VS Code. The terminal output is as follows:

```
PS C:\B.TECH\3.2> c::; cd 'c:\B.TECH\3.2'; & 'c:\Users\kamer\AppData\Local\Microsoft\WindowsApps\python3.12.exe' 'c:\Users\kamer\vscode\extensions\ms-python.debugpy-2025.10.0-win32-x64\bundled\libs\debugpy\launcher' '62401' '--' 'C:\B.TECH\3.2\TASK 5.py'  
Enter the temperature value: 37  
Enter the unit to convert from (C/F/K): C  
Enter the unit to convert to (C/F/K): F  
37.0°C = 98.60°F  
PS C:\B.TECH\3.2> |
```

The bottom status bar indicates 'Ln 37, Col 29', 'Spaces: 4', 'UTF-8', 'CRLF', 'Python', and '3.12.10 (Microsoft Store)'.

OBSERVATION: I observed that GitHub copilot can quickly generate working code for tasks such as login systems, loan approvals, Fibonacci functions, and job applicant scoring. However, the generated code sometimes contains issues like hardcoded values, lack of encryption, or biased decision logic. This shows that AI tools are helpful for faster coding but require human review for security, fairness, and correctness. GitHub Copilot is a fascinating tool to observe—especially in how it transforms the developer experience. Here's a breakdown of key observations across its functionality, impact, and adoption