LAB NAME : AI ASSISTED CODING

ROLL NO :2503A51L31

BRANCH : CSE

NAME : ch.abishek

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:

```python
# Function to calculate compound interest
def compound_interest(principal, rate, time, n):
    """
    Calculate the compound interest.

    Args:
        principal (float): The initial amount of money.
        rate (float): The annual interest rate (in percent).
        time (float): The time the money is invested for (in years).
        n (int): Number of times interest is compounded per year.

    Returns:
        float: The amount after interest.
    """
    amount = principal * (1 + rate / (100 * n)) ** (n * time)
    return amount

# Example usage:
if __name__ == "__main__":
    # principal = 1000, rate = 5%, time = 2 years, compounded quarterly (n=4)
    result = compound_interest(1000, 5, 2, 4)
    print(f"Compound interest amount: {result:.2f}")  # Output: Compound
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saian/anaconda3/pyth
Compound interest amount: 1104.49
```

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:

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:

```python
from statistics import mean, median, mode

def calculate_stats(numbers):
    avg = mean(numbers)
    med = median(numbers)
    try:
        mod = mode(numbers)
    except:
        mod = "No unique mode"
    return avg, med, mod

# Example usage:
nums = [1, 2, 2, 3, 4]
average, med, mod = calculate_stats(nums)
print(f"Average: {average}")
print(f"Median: {med}")
```

OUTPUT:

```
PS C:\Users\saian> & C:/Users/saia
Average: 2.4
Median: 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:

lab.3.3.py > ...

```python
def convert_to_binary(num: int) -> str:
    if num < 0:
        raise ValueError("Only non-negative integers are supported.")
    return bin(num)[2:]  # slice off the '0b' prefix


# Testing the function with example inputs
test_values = [5, 8, 15, 0, 1, 42, 255]

for value in test_values:
    print(f"Input: {value} -> Output: {convert_to_binary(value)}")
```

OUTPUT:

```
PS C:\Users\Dell\Desktop\AI> & C:/Users/Dell/anaconda3/python.e
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
```

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**.**

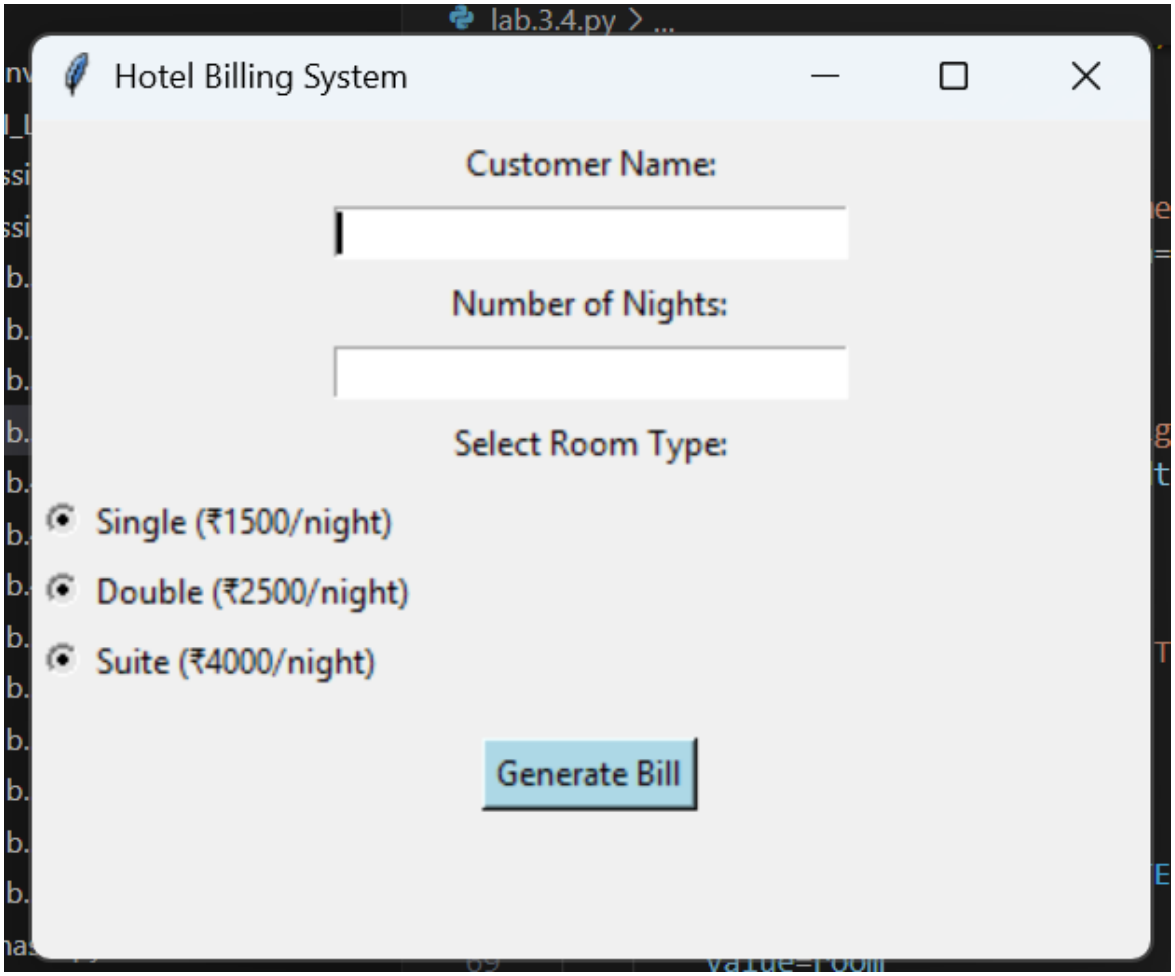
CODE:

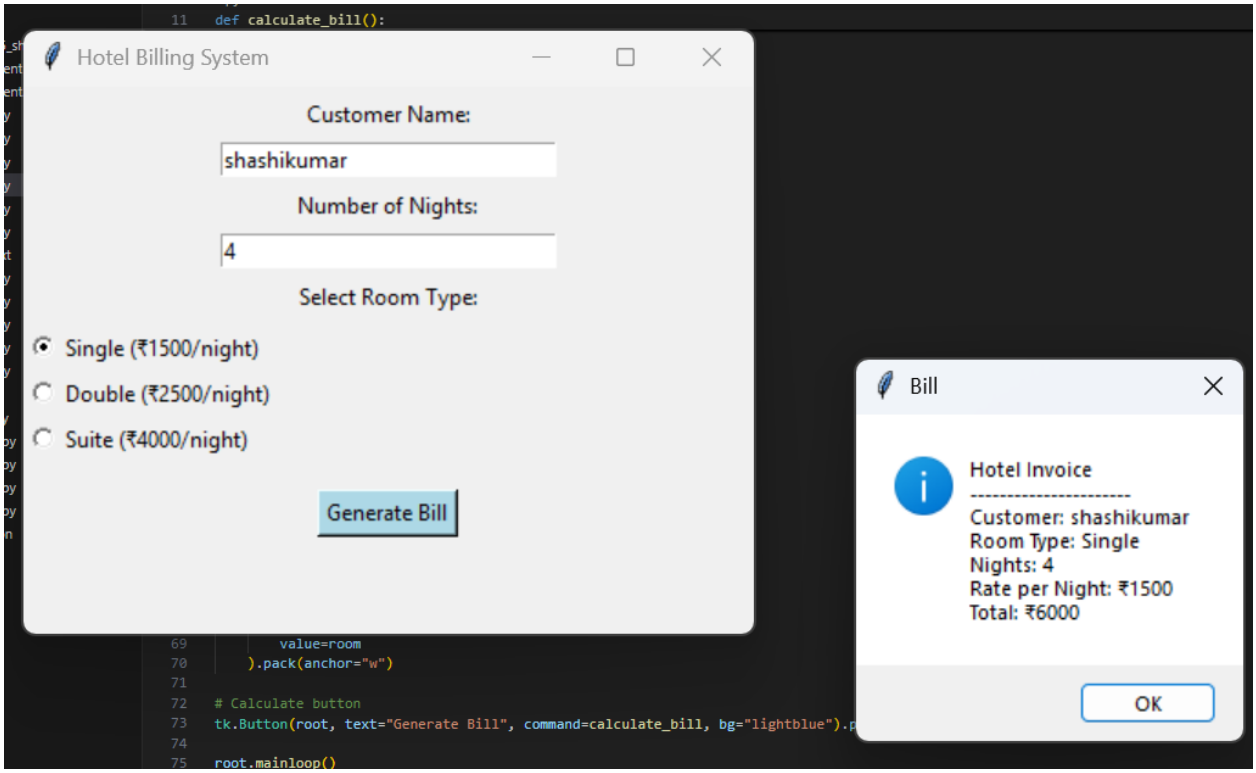
ab.3.4.py > ...

```python
import tkinter as tk
from tkinter import messagebox

# Prices for each room type per night
ROOM_RATES = {
    "Single": 1500,
    "Double": 2500,
    "Suite": 4000
}

def calculate_bill():
    try:
        name = entry_name.get().strip()
        nights = int(entry_nights.get())
        room_type = room_var.get()

        if not name:
            raise ValueError("Customer name is required")
        if nights <= 0:
            raise ValueError("Number of nights must be positive")
        if room_type not in ROOM_RATES:
            raise ValueError("Please select a room type")

        # Calculate total
        rate = ROOM_RATES[room_type]
        total = nights * rate

        # Display bill
        bill_text = (
            f"Hotel Invoice\n"
            f"----------------------\n"
            f"Customer: {name}\n"
            f"Room Type: {room_type}\n"
            f"Nights: {nights}\n"
            f"Rate per Night: ₹{rate}\n"
            f"Total: ₹{total}\n"
        )
        messagebox.showinfo("Bill", bill_text)

    except ValueError as ve:
        messagebox.showerror("Input Error", str(ve))
    except Exception as e:
```

```
    except ValueError as ve:
        messagebox.showerror("Input Error", str(ve))
    exce (module) messagebox
        messagebox.showerror("Error", f"Unexpected error: {e}")


# Main window
root = tk.Tk()
root.title("Hotel Billing System")
root.geometry("400x300")

# Customer Name
tk.Label(root, text="Customer Name:").pack(pady=5)
entry_name = tk.Entry(root, width=30)
entry_name.pack()

# Number of Nights
tk.Label(root, text="Number of Nights:").pack(pady=5)
entry_nights = tk.Entry(root, width=30)
entry_nights.pack()

# Room Type
tk.Label(root, text="Select Room Type:").pack(pady=5)
room_var = tk.StringVar(value="")
for room in ROOM_RATES:
    tk.Radiobutton(
        root,
        text=f"{room} (₹{ROOM_RATES[room]}/night)",
        variable=room_var,
        value=room
    ).pack(anchor="w")

# Calculate button
tk.Button(root, text="Generate Bill", command=calculate_bill, bg="lightblue").pack(pady=15)

root.mainloop()
```

OUTPUT:


lab.3.4.py > ...
Hotel Billing System
Customer Name:
Number of Nights:
Select Room Type:
Single (₹1500/night)
Double (₹2500/night)
Suite (₹4000/night)
Generate Bill



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:

lab.3.5.py > ...

```python
def convert_temperature(value: float, from_unit: str, to_unit: str) -> float:
    from_unit = from_unit.upper()
    to_unit = to_unit.upper()

    valid_units = ('C', 'F', 'K')
    if from_unit not in valid_units or to_unit not in valid_units:
        raise ValueError("Units must be 'C', 'F', or 'K'.")

    # Convert input to Celsius first
    if from_unit == 'C':
        celsius = value
    elif from_unit == 'F':
        celsius = (value - 32) * 5 / 9
    elif from_unit == 'K':
        celsius = value - 273.15

    # Convert Celsius to target unit
    if to_unit == 'C':
        return celsius
    elif to_unit == 'F':
        return (celsius * 9 / 5) + 32
    elif to_unit == 'K':
        return celsius + 273.15

# ===== Example Usage =====
if __name__ == "__main__":
    try:
        temp_value = float(input("Enter the temperature value: "))
        from_u = input("Enter the unit to convert from (C/F/K): ").strip()
        to_u = input("Enter the unit to convert to (C/F/K): ").strip()

        result = convert_temperature(temp_value, from_u, to_u)
        print(f"{temp_value}°{from_u.upper()} = {result:.2f}°{to_u.upper()}")

    except ValueError as e:
        print(f"Error: {e}")
```

OUTPUT:

```
PS C:\Users\Dell\Desktop\AI> & C:/Users/Dell/anaconda3/python.exe c:/Users/Dell/Desktop/AI/lab.3.5.py
Enter the temperature value: 36
Enter the unit to convert from (C/F/K): c
Enter the unit to convert to (C/F/K): f
36.0°C = 96.80°F
PS C:\Users\Dell\Desktop\AI>
```

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