

AIAC-3

Name: Gaje Harika

Ht.no: 2303A51612

Batch no:05

Task-01

Prompt : To generate a Python program that Reads: Previous Units (PU), Current Units (CU), Type of Customer, Calculates units consumed, Implements logic directly in the main program (no functions).

Code:

```
# Utility Bill Calculator

# Reads: Previous Units (PU), Current Units (CU), Type of Customer

# Calculates units consumed and bill amount

print("=" * 60)

print("UTILITY BILL CALCULATOR")

print("=" * 60)

# Read Previous Units (PU)

while True:

    try:

        pu = float(input("\nEnter Previous Units (PU): "))

        if pu < 0:

            print("Error: Previous Units cannot be negative. Please try again.")

            continue

        break

    except ValueError:

        print("Error: Please enter a valid number.")

# Read Current Units (CU)

while True:

    try:

        cu = float(input("Enter Current Units (CU): "))
```

```

    if cu < 0:
        print("Error: Current Units cannot be negative. Please try again.")
        continue

    if cu < pu:
        print("Error: Current Units cannot be less than Previous Units. Please try again.")
        continue

    break

except ValueError:
    print("Error: Please enter a valid number.")

# Read Type of Customer
print("\nCustomer Types:")
print("1. Domestic")
print("2. Commercial")
print("3. Industrial")

while True:
    try:
        customer_type_choice = input("\nEnter Customer Type (1/2/3): ").strip()

        if customer_type_choice == "1":
            customer_type = "Domestic"
            break

        elif customer_type_choice == "2":
            customer_type = "Commercial"
            break

        elif customer_type_choice == "3":
            customer_type = "Industrial"
            break

        else:
            print("Error: Please enter 1, 2, or 3.")

    except:
        print("Error: Invalid input. Please try again.")

# Calculate units consumed

```

```

units_consumed = cu - pu

# Display results

print("\n" + "=" * 60)

print("BILL CALCULATION")

print("=" * 60)

print(f"Previous Units (PU):   {pu:.2f} units")
print(f"Current Units (CU):   {cu:.2f} units")
print(f"Units Consumed:       {units_consumed:.2f} units")
print(f"Customer Type:        {customer_type}")

# Calculate bill based on customer type and units consumed

# Rate structure (example rates - can be modified)

if customer_type == "Domestic":
    if units_consumed <= 100:
        rate = 3.50
    elif units_consumed <= 200:
        rate = 4.50
    elif units_consumed <= 300:
        rate = 5.50
    else:
        rate = 6.50
elif customer_type == "Commercial":
    if units_consumed <= 100:
        rate = 5.00
    elif units_consumed <= 200:
        rate = 6.00
    elif units_consumed <= 300:
        rate = 7.00
    else:
        rate = 8.00
else: # Industrial
    if units_consumed <= 100:

```

```

        rate = 6.00

    elif units_consumed <= 200:

        rate = 7.50

    elif units_consumed <= 300:

        rate = 9.00

    else:

        rate = 10.50

# Calculate total bill

total_bill = units_consumed * rate

print(f"\nRate per unit:      ₹{rate:.2f}")

print(f"Total Bill Amount:    ₹{total_bill:.2f}")

print("=" * 60)

```

Output:

The screenshot displays a code editor with a Python script for calculating utility bills. The script includes comments explaining its purpose and steps, and a terminal window showing the program's execution with user input and formatted output.

```

Python unit consumption calculation
a short and clearly and briefly explanation about code.

Purpose: Calculate utility bill based on units consumed and customer type.

Main Steps:
1. Input Collection (Lines 9-55):
    - Reads Previous Units (PU) and Current Units (CU) with validation.
    - Reads Customer Type (1=Domestic, 2=Commercial, 3=Industrial).
    - Validates no negative values, CU > PU.
2. Calculation (Lines 57-100):
    - Units Consumed = CU - PU.
    - Determines rate per unit based on customer type and consumption fee.
    - Total Bill = Units Consumed * Rate.
3. Output (Lines 101-104):
    - Displays PU, CU, units consumed, customer type, rate, and total bill.

Key Features:
- Input validation with error handling.
- Tiered pricing (different rates for different consumption levels).
- Customer-specific rates (Domestic/Commercial/Industrial).

```

```

# Utility Bill Calculator
# Reads: Previous Units (PU), Current Units (CU), Type of Customer
# Calculates units consumed and bill amount

print("-" * 60)
print("UTILITY BILL CALCULATOR")
print("-" * 60)

# Read Previous Units (PU)
while True:
    try:
        pu = float(input("Enter Previous Units (PU): "))
        if pu < 0:
            print("Error: Previous Units cannot be negative, please try again.")
            continue
        break
    except ValueError:
        print("Error: Please enter a valid number.")

```

```

Enter Previous Units (PU): 12
Enter Current Units (CU): 34

Customer Types:
1. Domestic
2. Commercial
3. Industrial

Enter Customer Type (1/2/3): 1

=====
BILL CALCULATION
Previous Units (PU):    12.00 units

```

Enter Previous Units (PU): 12

Enter Current Units (CU): 34

Customer Types:

1. Domestic

2. Commercial

3. Industrial

3. Industrial

Enter Customer Type (1/2/3): 1

=====

BILL CALCULATION

=====

Previous Units (PU): 12.00 units

Current Units (CU): 34.00 units

Units Consumed: 22.00 units

Customer Type: Domestic

Rate per unit: ₹3.50

Total Bill Amount: ₹77.00

Explanation:

Purpose: Calculates utility bill based on units consumed and customer type.**Main Steps:**

1. Input Collection (Lines 9-55):

- Reads Previous Units (PU) and Current Units (CU) with validation
- Reads Customer Type (1=Domestic, 2=Commercial, 3=Industrial)
- Validates: no negative values, $CU \geq PU$

2. Calculation (Lines 57-100):

- $Units\ Consumed = CU - PU$
- Determines rate per unit based on customer type and consumption tier
- $Total\ Bill = Units\ Consumed \times Rate$

3. Output (Lines 60-104):

- Displays PU, CU, units consumed, customer type, rate, and total bill

Key Features:

- Input validation with error handling
- Tiered pricing (different rates for different consumption levels)
- Customer-specific rates (Domestic/Commercial/Industrial)
- All logic in main program (no functions)

Example: If PU=100, CU=250, Type=Domestic → Units=150 → Rate=₹4.50 → Bill=₹675.00

Task-02

Prompt: Add Energy Charge (EC) calculation with optimized conditional logic for Domestic, Commercial, and Industrial consumers. Simplify and improve code readability

Code:

class UtilityBillCalculator:

```
RATE_STRUCTURE = {  
    "Domestic": [  
        (0, 100, 3.50),  
        (101, 200, 4.50),  
        (201, 300, 5.50),  
        (301, float('inf'), 6.50)  
    ],  
    "Commercial": [  
        (0, 100, 5.00),  
        (101, 200, 6.00),  
        (201, 300, 7.00),  
        (301, float('inf'), 8.00)  
    ],  
    "Industrial": [  
        (0, 100, 6.00),  
        (101, 200, 7.50),  
        (201, 300, 9.00),  
        (301, float('inf'), 10.50)  
    ]  
}
```

```
def __init__(self, previous_units: float, current_units: float, customer_type: str):
```

```
    """
```

```
    Initialize the calculator with meter readings and customer type.
```

```
    Args:
```

```
        previous_units: Previous meter reading (PU)
```

```
        current_units: Current meter reading (CU)
```

```
        customer_type: Type of customer (Domestic, Commercial, Industrial)
```

```
    """
```

```
    self.previous_units = previous_units
```

```
    self.current_units = current_units
```

```
    self.customer_type = customer_type
```

```
    self.units_consumed = current_units - previous_units
```

```
    # Validate inputs
```

```
    self._validate_inputs()
```

```
def _validate_inputs(self):
```

```
    """Validate that inputs are logical and valid."""
```

```
    if self.previous_units < 0:
```

```
        raise ValueError("Previous Units cannot be negative")
```

```
    if self.current_units < 0:
```

```
        raise ValueError("Current Units cannot be negative")
```

```
    if self.current_units < self.previous_units:
```

```
        raise ValueError("Current Units cannot be less than Previous Units")
```

```
    if self.customer_type not in self.RATE_STRUCTURE:
```

```
        raise ValueError(f"Invalid customer type: {self.customer_type}")
```

```
def get_rate_per_unit(self) -> float:
```

```
    """
```

```
    Get the applicable rate per unit based on units consumed and customer type.
```

```
    Optimized conditional logic using slab-based lookup.
```

```
    Returns:
```

```

    Rate per unit in rupees
    """

    rate_slabs = self.RATE_STRUCTURE[self.customer_type]
    # Optimized: Direct lookup based on units consumed
    for min_units, max_units, rate in rate_slabs:
        if min_units <= self.units_consumed <= max_units:
            return rate

    # Fallback (should not reach here with proper structure)
    return rate_slabs[-1][2]

def calculate_energy_charges(self) -> float:
    rate_per_unit = self.get_rate_per_unit()
    energy_charges = self.units_consumed * rate_per_unit
    return energy_charges

def get_bill_details(self) -> dict:
    rate_per_unit = self.get_rate_per_unit()
    energy_charges = self.calculate_energy_charges()
    return {
        "previous_units": self.previous_units,
        "current_units": self.current_units,
        "units_consumed": self.units_consumed,
        "customer_type": self.customer_type,
        "rate_per_unit": rate_per_unit,
        "energy_charges": energy_charges,
        "total_bill": energy_charges
    }

def display_bill(self):
    """Display the bill in a formatted manner."""
    details = self.get_bill_details()
    print("\n" + "=" * 60)

```



```

print("BILL CALCULATION")

print("=" * 60)

print(f"Previous Units (PU):    {details['previous_units']:.2f} units")
print(f"Current Units (CU):    {details['current_units']:.2f} units")
print(f"Units Consumed:        {details['units_consumed']:.2f} units")
print(f"Customer Type:         {details['customer_type']}")
print(f"\nRate per unit:        ₹{details['rate_per_unit']:.2f}")
print(f"Energy Charges (EC):    ₹{details['energy_charges']:.2f}")
print(f"Total Bill Amount:      ₹{details['total_bill']:.2f}")

print("=" * 60)

def get_user_input() -> tuple[float, float, str]:

    print("=" * 60)

    print("UTILITY BILL CALCULATOR (Class-Based Implementation)")

    print("=" * 60)

    # Get Previous Units

    while True:

        try:

            pu = float(input("\nEnter Previous Units (PU): "))

            if pu < 0:

                print("Error: Previous Units cannot be negative. Please try again.")

                continue

            break

        except ValueError:

            print("Error: Please enter a valid number.")

    # Get Current Units

    while True:

        try:

            cu = float(input("Enter Current Units (CU): "))

            if cu < 0:

```

```

        print("Error: Current Units cannot be negative. Please try again.")
        continue
    if cu < pu:
        print("Error: Current Units cannot be less than Previous Units. Please try again.")
        continue
    break
except ValueError:
    print("Error: Please enter a valid number.")

# Get Customer Type
print("\nCustomer Types:")
print("1. Domestic")
print("2. Commercial")
print("3. Industrial")
customer_types = {"1": "Domestic", "2": "Commercial", "3": "Industrial"}
while True:
    choice = input("\nEnter Customer Type (1/2/3): ").strip()
    if choice in customer_types:
        return pu, cu, customer_types[choice]
    print("Error: Please enter 1, 2, or 3.")

def main():
    """Main program entry point."""
    try:
        # Get user input
        pu, cu, customer_type = get_user_input()

        # Create calculator instance
        calculator = UtilityBillCalculator(pu, cu, customer_type)

        # Display bill
        calculator.display_bill()
    except ValueError as e:

```

```
print(f"\nError: {e}")
```

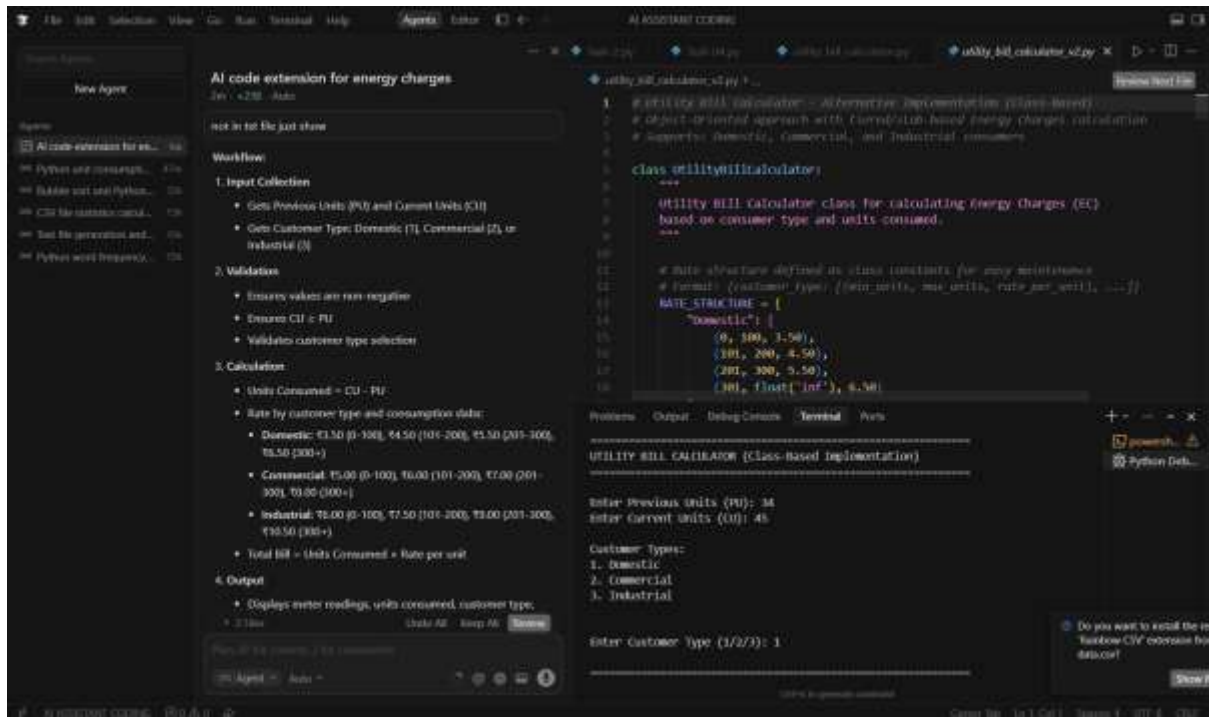
except Exception as e:

```
print(f"\nUnexpected error: {e}")
```

```
if __name__ == "__main__":
```

```
    main()
```

Output:



UTILITY BILL CALCULATOR (Class-Based Implementation)

Enter Previous Units (PU): 34

Enter Current Units (CU): 45

Customer Types:

1. Domestic

2. Commercial

3. Industrial

Enter Customer Type (1/2/3): 1

BILL CALCULATION

Previous Units (PU): 34.00 units

Current Units (CU): 45.00 units

Units Consumed: 11.00 units

Customer Type: Domestic

Rate per unit: ₹3.50

Energy Charges (EC): ₹38.50

Total Bill Amount: ₹38.50

Explanation:

Purpose: Calculates electricity bill from meter readings and customer type.

Workflow:

1. Input Collection

- Gets Previous Units (PU) and Current Units (CU)
- Gets Customer Type: Domestic (1), Commercial (2), or Industrial (3)

2. Validation

- Ensures values are non-negative
- Ensures $CU \geq PU$
- Validates customer type selection

3. Calculation

- Units Consumed = $CU - PU$
- Rate by customer type and consumption slabs:
 - **Domestic:** ₹3.50 (0-100), ₹4.50 (101-200), ₹5.50 (201-300), ₹6.50 (300+)
 - **Commercial:** ₹5.00 (0-100), ₹6.00 (101-200), ₹7.00 (201-300), ₹8.00 (300+)
 - **Industrial:** ₹6.00 (0-100), ₹7.50 (101-200), ₹9.00 (201-300), ₹10.50 (300+)
- Total Bill = Units Consumed × Rate per unit

4. Output

- Displays meter readings, units consumed, customer type, rate, and total bill

Task-03

Prompt: Develop a Python program with user-defined functions for calculating Energy Charges and Fixed Charges. Functions should accept inputs, return results, use simple logic

Code:

```

def calculate_energy_charges(units_consumed, customer_type):

    rate_structure = {

        "Domestic": [

            (0, 100, 3.50),    # First 100 units at ₹3.50 per unit

            (101, 200, 4.50), # Next 100 units (101-200) at ₹4.50 per unit

            (201, 300, 5.50), # Next 100 units (201-300) at ₹5.50 per unit

            (301, float('inf'), 6.50) # Above 300 units at ₹6.50 per unit

        ],

        "Commercial": [

            (0, 100, 5.00),    # First 100 units at ₹5.00 per unit

            (101, 200, 6.00), # Next 100 units (101-200) at ₹6.00 per unit

            (201, 300, 7.00), # Next 100 units (201-300) at ₹7.00 per unit

            (301, float('inf'), 8.00) # Above 300 units at ₹8.00 per unit

        ],

        "Industrial": [

            (0, 100, 6.00),    # First 100 units at ₹6.00 per unit

            (101, 200, 7.50), # Next 100 units (101-200) at ₹7.50 per unit

            (201, 300, 9.00), # Next 100 units (201-300) at ₹9.00 per unit

            (301, float('inf'), 10.50) # Above 300 units at ₹10.50 per unit

        ]

    }

    # Validate customer type

    if customer_type not in rate_structure:

        raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic, Commercial, or Industrial")

    # Get the applicable rate slabs for the customer type

    slabs = rate_structure[customer_type]

    # Find the applicable rate based on units consumed

    # The rate is determined by which consumption slab the total units fall into

    # All units are charged at the rate of the slab they fall into

```

```

for min_units, max_units, rate_per_unit in slabs:
    if min_units <= units_consumed <= max_units:
        # Units fall within this slab, calculate charges at this rate
        energy_charges = units_consumed * rate_per_unit
        return energy_charges

# Fallback: If consumption exceeds all defined slabs, use the highest rate
# This handles cases where consumption is very high
highest_rate = slabs[-1][2]
return units_consumed * highest_rate

def calculate_fixed_charges(customer_type, units_consumed):
    fixed_charges_structure = {
        "Domestic": {
            "low": 50.00,    # For consumption up to 100 units
            "medium": 75.00, # For consumption 101-200 units
            "high": 100.00,  # For consumption 201-300 units
            "very_high": 150.00 # For consumption above 300 units
        },
        "Commercial": {
            "low": 200.00,   # For consumption up to 100 units
            "medium": 300.00, # For consumption 101-200 units
            "high": 400.00,  # For consumption 201-300 units
            "very_high": 500.00 # For consumption above 300 units
        },
        "Industrial": {
            "low": 500.00,   # For consumption up to 100 units
            "medium": 750.00, # For consumption 101-200 units
            "high": 1000.00, # For consumption 201-300 units
            "very_high": 1500.00 # For consumption above 300 units
        }
    }

```

```

}

# Validate customer type

if customer_type not in fixed_charges_structure:

    raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic,
Commercial, or Industrial")

# Determine consumption category based on units consumed

if units_consumed <= 100:

    category = "low"

elif units_consumed <= 200:

    category = "medium"

elif units_consumed <= 300:

    category = "high"

else:

    category = "very_high"

# Get and return the fixed charges for this customer type and category
fixed_charges = fixed_charges_structure[customer_type][category]

return fixed_charges

def calculate_total_bill(previous_units, current_units, customer_type):

    # Validate inputs

    if previous_units < 0:

        raise ValueError("Previous Units cannot be negative")

    if current_units < 0:

        raise ValueError("Current Units cannot be negative")

    if current_units < previous_units:

        raise ValueError("Current Units cannot be less than Previous Units")

    # Calculate units consumed (difference between current and previous readings)

    units_consumed = current_units - previous_units

    # Calculate Energy Charges using the dedicated function

    energy_charges = calculate_energy_charges(units_consumed, customer_type)

    # Calculate Fixed Charges using the dedicated function

```

```

fixed_charges = calculate_fixed_charges(customer_type, units_consumed)
# Calculate total bill (Energy Charges + Fixed Charges)
total_bill = energy_charges + fixed_charges
# Return all calculated values as a dictionary
return {
    "previous_units": previous_units,
    "current_units": current_units,
    "units_consumed": units_consumed,
    "customer_type": customer_type,
    "energy_charges": energy_charges,
    "fixed_charges": fixed_charges,
    "total_bill": total_bill
}

```

```

def get_user_input():
    print("=" * 70)
    print("UTILITY BILL CALCULATOR - Function-Based Implementation")
    print("=" * 70)
    # Get Previous Units (PU) with validation
    while True:
        try:
            previous_units = float(input("\nEnter Previous Units (PU): "))
            if previous_units < 0:
                print("Error: Previous Units cannot be negative. Please try again.")
                continue
            break
        except ValueError:
            print("Error: Please enter a valid number.")
    # Get Current Units (CU) with validation
    while True:

```



```

try:
    current_units = float(input("Enter Current Units (CU): "))
    if current_units < 0:
        print("Error: Current Units cannot be negative. Please try again.")
        continue
    if current_units < previous_units:
        print("Error: Current Units cannot be less than Previous Units. Please try again.")
        continue
    break
except ValueError:
    print("Error: Please enter a valid number.")

# Get Customer Type with validation
print("\nCustomer Types:")
print("1. Domestic")
print("2. Commercial")
print("3. Industrial")
customer_types = {"1": "Domestic", "2": "Commercial", "3": "Industrial"}
while True:
    choice = input("\nEnter Customer Type (1/2/3): ").strip()
    if choice in customer_types:
        customer_type = customer_types[choice]
        break
    print("Error: Please enter 1, 2, or 3.")
return previous_units, current_units, customer_type

def display_bill(bill_details):
    print("\n" + "=" * 70)
    print("BILL CALCULATION RESULTS")
    print("=" * 70)
    print(f"Previous Units (PU):    {bill_details['previous_units']:.2f} units")

```

```

print(f"Current Units (CU):      {bill_details['current_units']:.2f} units")
print(f"Units Consumed:        {bill_details['units_consumed']:.2f} units")
print(f"Customer Type:         {bill_details['customer_type']}")
print("-" * 70)

print(f"Energy Charges (EC):     ₹{bill_details['energy_charges']:.2f}")
print(f"Fixed Charges (FC):      ₹{bill_details['fixed_charges']:.2f}")
print("-" * 70)

print(f"TOTAL BILL AMOUNT:       ₹{bill_details['total_bill']:.2f}")
print("=" * 70)

```

```
def main():
```

```
    try:
```

```
        # Step 1: Get user input
```

```
        previous_units, current_units, customer_type = get_user_input()
```

```
        # Step 2: Calculate bill using the calculate_total_bill function
```

```
        # This function internally calls calculate_energy_charges and calculate_fixed_charges
```

```
        bill_details = calculate_total_bill(previous_units, current_units, customer_type)
```

```
        # Step 3: Display the calculated bill
```

```
        display_bill(bill_details)
```

```
    except ValueError as e:
```

```
        print(f"\nError: {e}")
```

```
    except Exception as e:
```

```
        print(f"\nUnexpected error: {e}")
```

```
# Program entry point
```

```
if __name__ == "__main__":
```

```
    main()
```

Output:

UTILITY BILL CALCULATOR - Function-Based Implementation

Enter Previous Units (PU): 45

Enter Current Units (CU): 56

Customer Types:

1. Domestic
2. Commercial
3. Industrial

Enter Customer Type (1/2/3): 2

BILL CALCULATION RESULTS

Previous Units (PU): 45.00 units
Current Units (CU): 56.00 units
Units Consumed: 11.00 units
Customer Type: Commercial
Energy Charges (EC): ₹55.00
Fixed Charges (FC): ₹200.00
TOTAL BILL AMOUNT: ₹255.00

The screenshot shows a VS Code editor with a Python file named `utility_bill_calculator.py`. The script defines a `calculate_energy_charges` function that takes `units_consumed` and `customer_type` as arguments. It uses a nested dictionary `rate_structure` to calculate energy charges based on consumption ranges and customer type. The function also includes comments explaining the logic and the rate structure.

```
def calculate_energy_charges(units_consumed, customer_type):  
    rate_structure = {  
        "domestic": {  
            (0, 100, 3.50), # First 100 units at ₹3.50 per unit  
            (101, 200, 4.50), # Next 100 units (101-200) at ₹4.50 per unit  
            (201, 300, 5.50), # Next 100 units (201-300) at ₹5.50 per unit  
            (301, float('inf'), 6.50) # Above 300 units at ₹6.50 per unit  
        },  
        "commercial": {  
            (0, 100, 5.00), # First 100 units at ₹5.00 per unit  
            (101, 200, 6.00), # Next 100 units (101-200) at ₹6.00 per unit  
            (201, 300, 7.00), # Next 100 units (201-300) at ₹7.00 per unit  
            (301, float('inf'), 8.00) # Above 300 units at ₹8.00 per unit  
        },  
        "industrial": {  
            (0, 100, 6.00), # First 100 units at ₹6.00 per unit  
            (101, 200, 7.50), # Next 100 units (101-200) at ₹7.50 per unit  
            (201, 300, 9.00), # Next 100 units (201-300) at ₹9.00 per unit  
            (301, float('inf'), 10.00) # Above 300 units at ₹10.00 per unit  
        }  
    }  
    # Calculate energy charges  
    rate_range = list(rate_structure[customer_type].keys())  
    rate_start, rate_end, rate_per_unit = rate_range[0]  
    units_consumed = max(0, units_consumed)  
    energy_charges = 0  
    for i in range(0, units_consumed, rate_end - rate_start + 1):  
        rate_start, rate_end, rate_per_unit = rate_range[i]  
        energy_charges += (rate_end - rate_start + 1) * rate_per_unit  
    return energy_charges
```

The terminal output shows the results of the calculation:

```
UTILITY BILL CALCULATOR - function based Implementation  
  
Enter Previous units (PU): 45  
Enter Current Units (CU): 56  
  
Customer Types:  
1. Domestic  
2. Commercial  
3. Industrial  
  
Enter Customer Type (1/2/3): 2  
  
BILL CALCULATION RESULTS  
  
Previous Units (PU): 45.00 units  
Current Units (CU): 56.00 units  
Units Consumed: 11.00 units  
Customer Type: Commercial  
Energy Charges (EC): ₹55.00  
Fixed Charges (FC): ₹200.00  
TOTAL BILL AMOUNT: ₹255.00
```

Explanation:

1. User enters previous units, current units, and customer type
2. Program calculates units consumed = current - previous
3. Energy Charges = units × rate (based on consumption slab)
4. Fixed Charges = fixed amount (based on customer type and consumption category)
5. Total Bill = Energy Charges + Fixed Charges
6. Results are displayed

Task-04

Prompt:

Extend the program to calculate Fixed Charges (FC), Customer Charges (CC), and Electricity Duty. Add electricity duty calculation, improve billing accuracy, and keep the code simple and readable.

Code:

```
def calculate_energy_charges(units_consumed, customer_type):  
    rate_structure = {  
        "Domestic": [  
            (0, 100, 3.50),    # First 100 units at ₹3.50 per unit  
            (101, 200, 4.50), # Next 100 units (101-200) at ₹4.50 per unit  
            (201, 300, 5.50), # Next 100 units (201-300) at ₹5.50 per unit  
            (301, float('inf'), 6.50) # Above 300 units at ₹6.50 per unit  
        ],  
        "Commercial": [  
            (0, 100, 5.00),    # First 100 units at ₹5.00 per unit  
            (101, 200, 6.00), # Next 100 units (101-200) at ₹6.00 per unit  
            (201, 300, 7.00), # Next 100 units (201-300) at ₹7.00 per unit  
            (301, float('inf'), 8.00) # Above 300 units at ₹8.00 per unit  
        ],  
    }
```

```

"Industrial": [
    (0, 100, 6.00),    # First 100 units at ₹6.00 per unit
    (101, 200, 7.50), # Next 100 units (101-200) at ₹7.50 per unit
    (201, 300, 9.00), # Next 100 units (201-300) at ₹9.00 per unit
    (301, float('inf'), 10.50) # Above 300 units at ₹10.50 per unit
]
}

# Validate customer type
if customer_type not in rate_structure:
    raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic, Commercial, or Industrial")

# Get the applicable rate slabs for the customer type
slabs = rate_structure[customer_type]

# Find the applicable rate based on units consumed
# The rate is determined by which consumption slab the total units fall into
# All units are charged at the rate of the slab they fall into
for min_units, max_units, rate_per_unit in slabs:
    if min_units <= units_consumed <= max_units:
        # Units fall within this slab, calculate charges at this rate
        energy_charges = units_consumed * rate_per_unit
        return energy_charges

# Fallback: If consumption exceeds all defined slabs, use the highest rate
# This handles cases where consumption is very high
highest_rate = slabs[-1][2]
return units_consumed * highest_rate

```

```

def calculate_fixed_charges(customer_type, units_consumed):

    # Define fixed charges structure

    # Fixed charges may vary by customer type and consumption category
    fixed_charges_structure = {

        "Domestic": {

            "low": 50.00,    # For consumption up to 100 units

            "medium": 75.00, # For consumption 101-200 units

            "high": 100.00,  # For consumption 201-300 units

            "very_high": 150.00 # For consumption above 300 units

        },

        "Commercial": {

            "low": 200.00,    # For consumption up to 100 units

            "medium": 300.00, # For consumption 101-200 units

            "high": 400.00,   # For consumption 201-300 units

            "very_high": 500.00 # For consumption above 300 units

        },

        "Industrial": {

            "low": 500.00,    # For consumption up to 100 units

            "medium": 750.00, # For consumption 101-200 units

            "high": 1000.00,  # For consumption 201-300 units

            "very_high": 1500.00 # For consumption above 300 units

        }

    }

    # Validate customer type

    if customer_type not in fixed_charges_structure:

        raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic, Commercial, or Industrial")

    # Determine consumption category based on units consumed

```

```
if units_consumed <= 100:
```

```
    category = "low"
```

```
elif units_consumed <= 200:
```

```
    category = "medium"
```

```
elif units_consumed <= 300:
```

```
    category = "high"
```

```
else:
```

```
    category = "very_high"
```

```
# Get and return the fixed charges for this customer type and category
```

```
fixed_charges = fixed_charges_structure[customer_type][category]
```

```
return fixed_charges
```

```
def calculate_customer_charges(customer_type, units_consumed):
```

```
    # Define customer charges structure
```

```
    # Customer charges vary by customer type and consumption level
```

```
    customer_charges_structure = {
```

```
        "Domestic": {
```

```
            "low": 25.00,    # For consumption up to 100 units
```

```
            "medium": 35.00, # For consumption 101-200 units
```

```
            "high": 50.00,   # For consumption 201-300 units
```

```
            "very_high": 75.00 # For consumption above 300 units
```

```
        },
```

```
        "Commercial": {
```

```
            "low": 100.00,   # For consumption up to 100 units
```

```
            "medium": 150.00, # For consumption 101-200 units
```

```
            "high": 200.00,  # For consumption 201-300 units
```

```
            "very_high": 300.00 # For consumption above 300 units
```

```
        },
```

```
"Industrial": {  
    "low": 250.00,  # For consumption up to 100 units  
    "medium": 400.00, # For consumption 101-200 units  
    "high": 600.00,  # For consumption 201-300 units  
    "very_high": 900.00 # For consumption above 300 units  
}  
}
```

Validate customer type

if customer_type not in customer_charges_structure:

*raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic,
Commercial, or Industrial")*

Determine consumption category based on units consumed

if units_consumed <= 100:

category = "low"

elif units_consumed <= 200:

category = "medium"

elif units_consumed <= 300:

category = "high"

else:

category = "very_high"

Get and return the customer charges for this customer type and category

customer_charges = customer_charges_structure[customer_type][category]

return customer_charges

def calculate_electricity_duty(energy_charges, customer_type):

Define electricity duty percentage by customer type

ED is calculated as a percentage of Energy Charges


```
duty_percentage = {  
    "Domestic": 5.0,    # 5% of EC for Domestic customers  
    "Commercial": 8.0, # 8% of EC for Commercial customers  
    "Industrial": 10.0 # 10% of EC for Industrial customers  
}
```

```
# Validate customer type
```

```
if customer_type not in duty_percentage:
```

```
    raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic,  
Commercial, or Industrial")
```

```
# Validate energy charges
```

```
if energy_charges < 0:
```

```
    raise ValueError("Energy Charges cannot be negative")
```

```
# Calculate electricity duty as percentage of energy charges
```

```
duty_percent = duty_percentage[customer_type]
```

```
electricity_duty = energy_charges * (duty_percent / 100.0)
```

```
return electricity_duty
```

```
def calculate_total_bill(previous_units, current_units, customer_type):
```

```
    # Validate inputs
```

```
    if previous_units < 0:
```

```
        raise ValueError("Previous Units cannot be negative")
```

```
    if current_units < 0:
```

```
        raise ValueError("Current Units cannot be negative")
```

```
    if current_units < previous_units:
```

```
        raise ValueError("Current Units cannot be less than Previous Units")
```

Calculate units consumed (difference between current and previous readings)

`units_consumed = current_units - previous_units`

Calculate Energy Charges using the dedicated function

`energy_charges = calculate_energy_charges(units_consumed, customer_type)`

Calculate Fixed Charges using the dedicated function

`fixed_charges = calculate_fixed_charges(customer_type, units_consumed)`

Calculate Customer Charges using the dedicated function

`customer_charges = calculate_customer_charges(customer_type, units_consumed)`

Calculate Electricity Duty as percentage of Energy Charges

`electricity_duty = calculate_electricity_duty(energy_charges, customer_type)`

Calculate total bill (EC + FC + CC + ED)

`total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty`

Return all calculated values as a dictionary

`return {`

`"previous_units": previous_units,`

`"current_units": current_units,`

`"units_consumed": units_consumed,`

`"customer_type": customer_type,`

`"energy_charges": energy_charges,`

`"fixed_charges": fixed_charges,`

`"customer_charges": customer_charges,`

`"electricity_duty": electricity_duty,`

`"total_bill": total_bill`

```
}
```

```
def get_user_input():
```

```
    print("=" * 70)
```

```
    print("UTILITY BILL CALCULATOR - Extended Version")
```

```
    print("Calculates: EC, FC, CC, and ED")
```

```
    print("=" * 70)
```

```
# Get Previous Units (PU) with validation
```

```
while True:
```

```
    try:
```

```
        previous_units = float(input("\nEnter Previous Units (PU): "))
```

```
        if previous_units < 0:
```

```
            print("Error: Previous Units cannot be negative. Please try again.")
```

```
            continue
```

```
        break
```

```
    except ValueError:
```

```
        print("Error: Please enter a valid number.")
```

```
# Get Current Units (CU) with validation
```

```
while True:
```

```
    try:
```

```
        current_units = float(input("Enter Current Units (CU): "))
```

```
        if current_units < 0:
```

```
            print("Error: Current Units cannot be negative. Please try again.")
```

```
            continue
```

```
        if current_units < previous_units:
```

```
            print("Error: Current Units cannot be less than Previous Units. Please try again.")
```

```
            continue
```

```

        break

    except ValueError:

        print("Error: Please enter a valid number.")


# Get Customer Type with validation
print("\nCustomer Types:")
print("1. Domestic")
print("2. Commercial")
print("3. Industrial")

customer_types = {"1": "Domestic", "2": "Commercial", "3": "Industrial"}

while True:

    choice = input("\nEnter Customer Type (1/2/3): ").strip()

    if choice in customer_types:

        customer_type = customer_types[choice]

        break

    print("Error: Please enter 1, 2, or 3.")

return previous_units, current_units, customer_type


def display_bill(bill_details):
    """
    Display the calculated bill in a formatted, user-friendly manner.

    Args:
        bill_details (dict): Dictionary containing bill calculation results
    """
    print("\n" + "=" * 70)

```

```

print("BILL CALCULATION RESULTS")

print("=" * 70)

print(f"Previous Units (PU):    {bill_details['previous_units']:.2f} units")
print(f"Current Units (CU):     {bill_details['current_units']:.2f} units")
print(f"Units Consumed:          {bill_details['units_consumed']:.2f} units")
print(f"Customer Type:           {bill_details['customer_type']}")

print("-" * 70)

print("CHARGE BREAKDOWN:")

print(f" Energy Charges (EC):     ₹{bill_details['energy_charges']:.2f}")
print(f" Fixed Charges (FC):      ₹{bill_details['fixed_charges']:.2f}")
print(f" Customer Charges (CC):    ₹{bill_details['customer_charges']:.2f}")
print(f" Electricity Duty (ED):    ₹{bill_details['electricity_duty']:.2f}")

print("-" * 70)

print(f"TOTAL BILL AMOUNT:        ₹{bill_details['total_bill']:.2f}")

print("=" * 70)

```

def **main**():

try:

Step 1: Get user input

previous_units, current_units, customer_type = get_user_input()

bill_details = calculate_total_bill(previous_units, current_units, customer_type)

Step 3: Display the calculated bill

display_bill(bill_details)

except ValueError as e:

print(f"\nError: {e}")

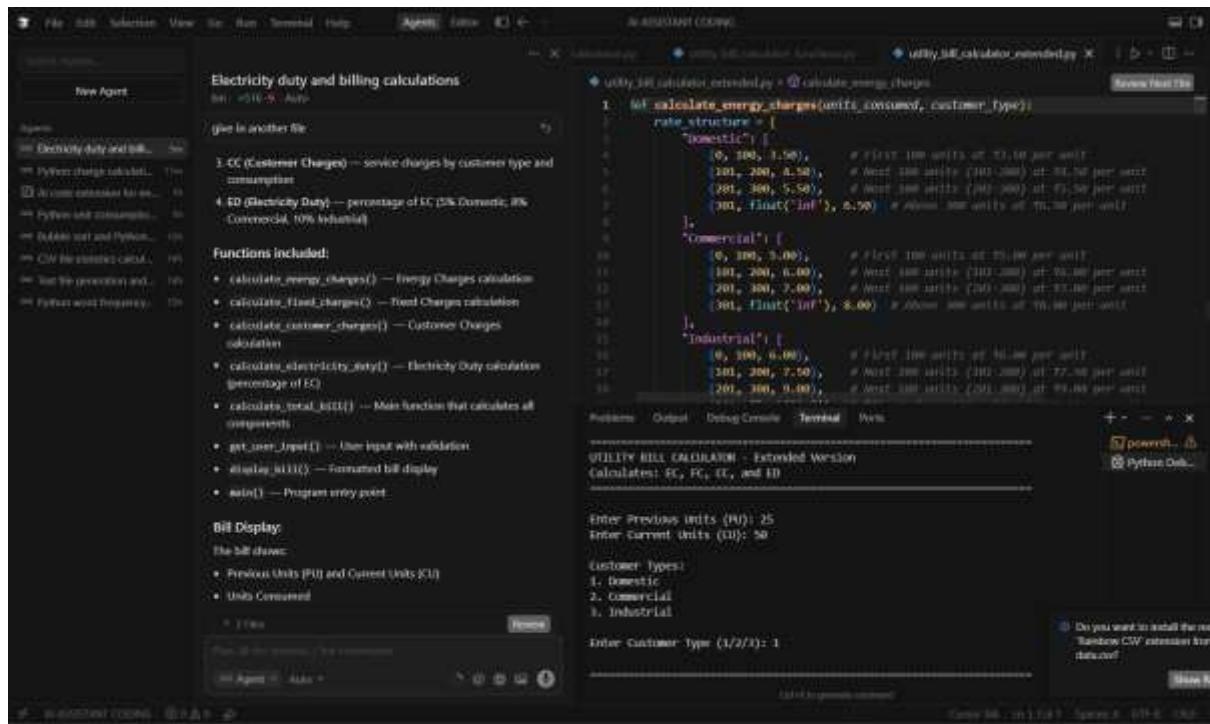
except Exception as e:

print(f"\nUnexpected error: {e}")

Program entry point

```
if __name__ == "__main__":  
    main()
```

Output:



UTILITY BILL CALCULATOR - Extended Version

Calculates: EC, FC, CC, and ED

Enter Previous Units (PU): 25

Enter Current Units (CU): 50

Customer Types:

1. Domestic

2. Commercial

3. Industrial

Enter Customer Type (1/2/3): 1

BILL CALCULATION RESULTS

Previous Units (PU): 25.00 units

Current Units (CU): 50.00 units

Units Consumed: 25.00 units

Customer Type: Domestic

CHARGE BREAKDOWN:

Energy Charges (EC): ₹87.50

Fixed Charges (FC): ₹50.00

Customer Charges (CC): ₹25.00

Electricity Duty (ED): ₹4.38

TOTAL BILL AMOUNT: ₹166.88

Explanation:

- `calculate_energy_charges()` — Energy Charges calculation
- `calculate_fixed_charges()` — Fixed Charges calculation
- `calculate_customer_charges()` — Customer Charges calculation
- `calculate_electricity_duty()` — Electricity Duty calculation (percentage of EC)
- `calculate_total_bill()` — Main function that calculates all components
- `get_user_input()` — User input with validation
- `display_bill()` — Formatted bill display
- `main()` — Program entry point

Task-05

Prompt: Develop a Python application to calculate and display Energy Charges (EC), Fixed Charges (FC), Customer Charges (CC), Electricity Duty (ED), and the Total Bill (EC + FC + CC + ED).

Code:

```
def calculate_energy_charges(units_consumed, customer_type):  
    rate_structure = {  
        "Domestic": [  
            (0, 100, 3.50),    # First 100 units at ₹3.50 per unit  
            (101, 200, 4.50),  # Next 100 units (101-200) at ₹4.50 per unit  
            (201, 300, 5.50),  # Next 100 units (201-300) at ₹5.50 per unit  
            (301, float('inf'), 6.50) # Above 300 units at ₹6.50 per unit
```

```

],
"Commercial": [
    (0, 100, 5.00),    # First 100 units at ₹5.00 per unit
    (101, 200, 6.00), # Next 100 units (101-200) at ₹6.00 per unit
    (201, 300, 7.00), # Next 100 units (201-300) at ₹7.00 per unit
    (301, float('inf'), 8.00) # Above 300 units at ₹8.00 per unit
],
"Industrial": [
    (0, 100, 6.00),    # First 100 units at ₹6.00 per unit
    (101, 200, 7.50), # Next 100 units (101-200) at ₹7.50 per unit
    (201, 300, 9.00), # Next 100 units (201-300) at ₹9.00 per unit
    (301, float('inf'), 10.50) # Above 300 units at ₹10.50 per unit
]
}

# Validate customer type
if customer_type not in rate_structure:
    raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic, Commercial, or Industrial")

# Validate units consumed
if units_consumed < 0:
    raise ValueError("Units consumed cannot be negative")

# Get the applicable rate slabs for the customer type
slabs = rate_structure[customer_type]

# Find the applicable rate based on units consumed
for min_units, max_units, rate_per_unit in slabs:
    if min_units <= units_consumed <= max_units:
        # Units fall within this slab, calculate charges at this rate
        energy_charges = units_consumed * rate_per_unit
    return energy_charges

```



```

# Fallback: If consumption exceeds all defined slabs, use the highest rate
highest_rate = slabs[-1][2]

return units_consumed * highest_rate

def calculate_fixed_charges(customer_type, units_consumed):
    fixed_charges_structure = {
        "Domestic": {
            "low": 50.00,    # For consumption up to 100 units
            "medium": 75.00, # For consumption 101-200 units
            "high": 100.00,  # For consumption 201-300 units
            "very_high": 150.00 # For consumption above 300 units
        },
        "Commercial": {
            "low": 200.00,   # For consumption up to 100 units
            "medium": 300.00, # For consumption 101-200 units
            "high": 400.00,  # For consumption 201-300 units
            "very_high": 500.00 # For consumption above 300 units
        },
        "Industrial": {
            "low": 500.00,   # For consumption up to 100 units
            "medium": 750.00, # For consumption 101-200 units
            "high": 1000.00, # For consumption 201-300 units
            "very_high": 1500.00 # For consumption above 300 units
        }
    }

    # Validate customer type
    if customer_type not in fixed_charges_structure:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic, Commercial, or Industrial")

    # Determine consumption category based on units consumed
    if units_consumed <= 100:

```

```

    category = "low"
elif units_consumed <= 200:
    category = "medium"
elif units_consumed <= 300:
    category = "high"
else:
    category = "very_high"

# Get and return the fixed charges for this customer type and category
fixed_charges = fixed_charges_structure[customer_type][category]
return fixed_charges

def calculate_customer_charges(customer_type, units_consumed):
    customer_charges_structure = {
        "Domestic": {
            "low": 25.00,    # For consumption up to 100 units
            "medium": 35.00, # For consumption 101-200 units
            "high": 50.00,   # For consumption 201-300 units
            "very_high": 75.00 # For consumption above 300 units
        },
        "Commercial": {
            "low": 100.00,   # For consumption up to 100 units
            "medium": 150.00, # For consumption 101-200 units
            "high": 200.00,  # For consumption 201-300 units
            "very_high": 300.00 # For consumption above 300 units
        },
        "Industrial": {
            "low": 250.00,   # For consumption up to 100 units
            "medium": 400.00, # For consumption 101-200 units
            "high": 600.00,  # For consumption 201-300 units
            "very_high": 900.00 # For consumption above 300 units
        }
    }

```

```

    }
}

# Validate customer type

if customer_type not in customer_charges_structure:

    raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic,
Commercial, or Industrial")

# Determine consumption category based on units consumed

if units_consumed <= 100:

    category = "low"

elif units_consumed <= 200:

    category = "medium"

elif units_consumed <= 300:

    category = "high"

else:

    category = "very_high"

# Get and return the customer charges for this customer type and category

customer_charges = customer_charges_structure[customer_type][category]

return customer_charges

def calculate_electricity_duty(energy_charges, customer_type):

    duty_percentage = {

        "Domestic": 5.0,    # 5% of EC for Domestic customers

        "Commercial": 8.0,  # 8% of EC for Commercial customers

        "Industrial": 10.0  # 10% of EC for Industrial customers

    }

    # Validate customer type

    if customer_type not in duty_percentage:

        raise ValueError(f"Invalid customer type: {customer_type}. Must be Domestic,
Commercial, or Industrial")

    # Validate energy charges

    if energy_charges < 0:

```

```

        raise ValueError("Energy Charges cannot be negative")

    # Calculate electricity duty as percentage of energy charges
    duty_percent = duty_percentage[customer_type]
    electricity_duty = energy_charges * (duty_percent / 100.0)
    return electricity_duty

def calculate_total_bill(previous_units, current_units, customer_type):
    # Validate inputs

    if previous_units < 0:
        raise ValueError("Previous Units cannot be negative")

    if current_units < 0:
        raise ValueError("Current Units cannot be negative")

    if current_units < previous_units:
        raise ValueError("Current Units cannot be less than Previous Units")

    # Calculate units consumed (difference between current and previous readings)
    units_consumed = current_units - previous_units

    # Calculate Energy Charges (EC)
    energy_charges = calculate_energy_charges(units_consumed, customer_type)

    # Calculate Fixed Charges (FC)
    fixed_charges = calculate_fixed_charges(customer_type, units_consumed)

    # Calculate Customer Charges (CC)
    customer_charges = calculate_customer_charges(customer_type, units_consumed)

    # Calculate Electricity Duty (ED) as percentage of Energy Charges
    electricity_duty = calculate_electricity_duty(energy_charges, customer_type)

    # Calculate total bill (EC + FC + CC + ED)
    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

    # Return all calculated values as a dictionary
    return {
        "previous_units": previous_units,
        "current_units": current_units,

```

```
"units_consumed": units_consumed,  
"customer_type": customer_type,  
"energy_charges": energy_charges,  
"fixed_charges": fixed_charges,  
"customer_charges": customer_charges,  
"electricity_duty": electricity_duty,  
"total_bill": total_bill  
}
```

```
def get_user_input():  
    print("=" * 70)  
    print("ELECTRICITY BILL CALCULATOR")  
    print("Calculates: EC, FC, CC, ED, and Total Bill")  
    print("=" * 70)  
    # Get Previous Units (PU) with validation  
    while True:  
        try:  
            previous_units = float(input("\nEnter Previous Units (PU): "))  
            if previous_units < 0:  
                print("Error: Previous Units cannot be negative. Please try again.")  
                continue  
            break  
        except ValueError:  
            print("Error: Please enter a valid number.")  
    # Get Current Units (CU) with validation  
    while True:  
        try:  
            current_units = float(input("Enter Current Units (CU): "))  
            if current_units < 0:
```

```

        print("Error: Current Units cannot be negative. Please try again.")
        continue

    if current_units < previous_units:
        print("Error: Current Units cannot be less than Previous Units. Please try again.")
        continue

    break

except ValueError:
    print("Error: Please enter a valid number.")

# Get Customer Type with validation
print("\nCustomer Types:")
print("1. Domestic")
print("2. Commercial")
print("3. Industrial")
customer_types = {"1": "Domestic", "2": "Commercial", "3": "Industrial"}

while True:
    choice = input("\nEnter Customer Type (1/2/3): ").strip()

    if choice in customer_types:
        customer_type = customer_types[choice]
        break

    print("Error: Please enter 1, 2, or 3.")

return previous_units, current_units, customer_type

def display_bill(bill_details):
    print("\n" + "=" * 70)
    print("ELECTRICITY BILL STATEMENT")
    print("=" * 70)

    print(f"Previous Units (PU):    {bill_details['previous_units']:.2f} units")
    print(f"Current Units (CU):      {bill_details['current_units']:.2f} units")
    print(f"Units Consumed:          {bill_details['units_consumed']:.2f} units")
    print(f"Customer Type:           {bill_details['customer_type']}")

```

```

print("-" * 70)

print("CHARGE BREAKDOWN:")

print(f" Energy Charges (EC):    ₹{bill_details['energy_charges']:.2f}")
print(f" Fixed Charges (FC):    ₹{bill_details['fixed_charges']:.2f}")
print(f" Customer Charges (CC):  ₹{bill_details['customer_charges']:.2f}")
print(f" Electricity Duty (ED):  ₹{bill_details['electricity_duty']:.2f}")

print("-" * 70)

print(f"TOTAL BILL AMOUNT:      ₹{bill_details['total_bill']:.2f}")

print("=" * 70)

def main():

    try:

        # Step 1: Get user input

        previous_units, current_units, customer_type = get_user_input()

        # Step 2: Calculate bill with all components

        bill_details = calculate_total_bill(previous_units, current_units, customer_type)

        # Step 3: Display the calculated bill

        display_bill(bill_details)

    except ValueError as e:

        print(f"\nError: {e}")

    except Exception as e:

        print(f"\nUnexpected error: {e}")

if __name__ == "__main__":

    main()

```

Output:

ELECTRICITY BILL STATEMENT

Previous Units (PU):	56.00 units
Current Units (CU):	56.00 units
Units Consumed:	0.00 units
Customer Type:	Domestic

CHARGE BREAKDOWN:

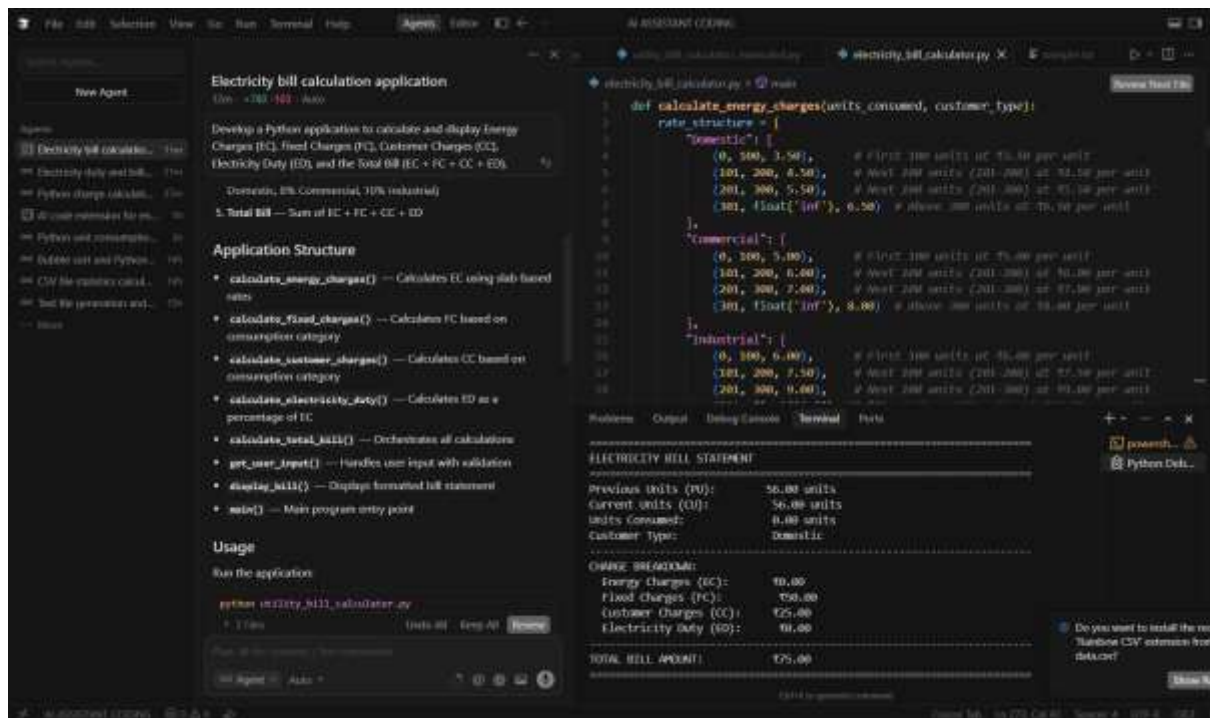
Energy Charges (EC): ₹0.00

Fixed Charges (FC): ₹50.00

Customer Charges (CC): ₹25.00

Electricity Duty (ED): ₹0.00

TOTAL BILL AMOUNT: ₹75.00



Explanation:

- `calculate_energy_charges()` — Calculates EC using slab-based rates
- `calculate_fixed_charges()` — Calculates FC based on consumption category
- `calculate_customer_charges()` — Calculates CC based on consumption category
- `calculate_electricity_duty()` — Calculates ED as a percentage of EC
- `calculate_total_bill()` — Orchestrates all calculations
- `get_user_input()` — Handles user input with validation
- `display_bill()` — Displays formatted bill statement
- `main()` — Main program entry point

