TASK-1: AI-Generated Logic for Reading Consumer Details

PROMPT:

An electricity billing system must collect accurate consumer data.

generate a Python program that:

• Reads:

o Previous Units (PU)

o Current Units (CU)

o Type of Customer

• Calculates units consumed

• Implements logic directly in the main program (no functions)

EXPLAINATION:

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 ≥ 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 × 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

CODE:

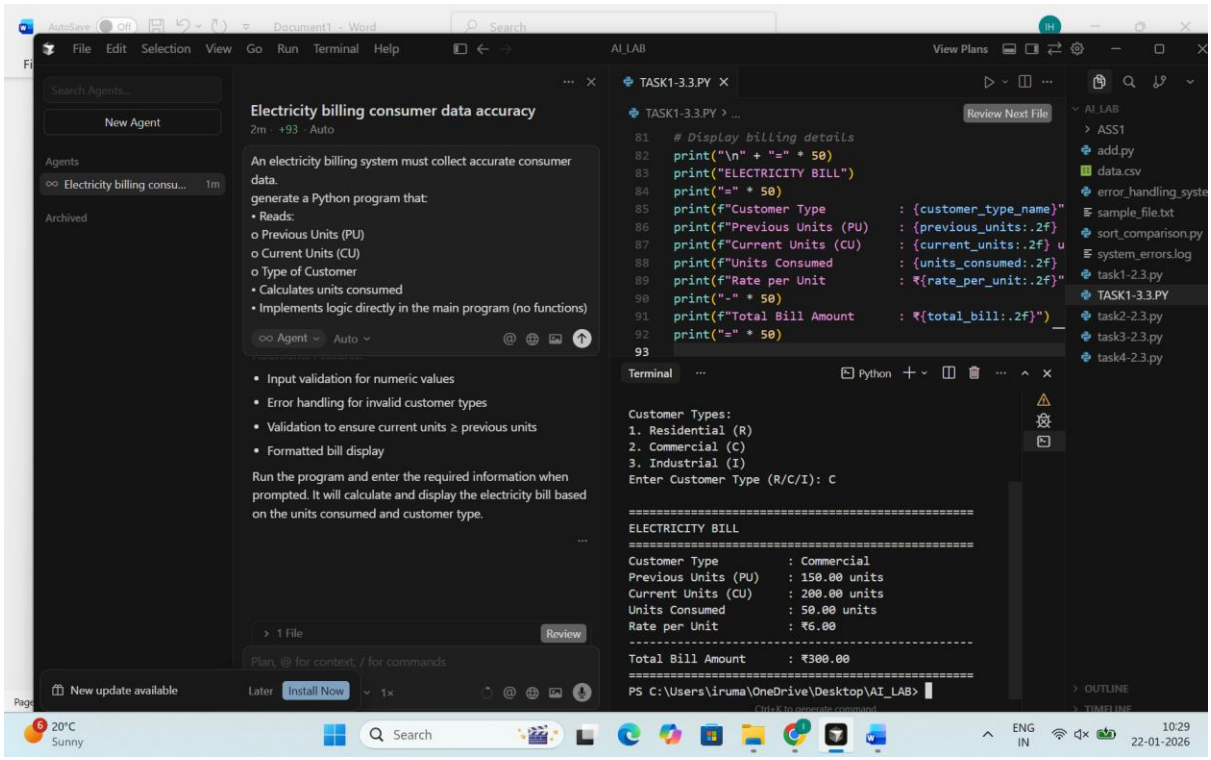
```
print("=" * 50)
print("ELECTRICITY BILLING SYSTEM")
```

```
print("=" * 50)
try:
    previous_units = float(input("Enter Previous Units (PU): "))
except ValueError:
    print("Error: Please enter a valid number for Previous Units")
    previous_units = 0.0
try:
    current_units = float(input("Enter Current Units (CU): "))
except ValueError:
    print("Error: Please enter a valid number for Current Units")
    current_units = 0.0print("\nCustomer Types:")
print("1. Residential (R)")
print("2. Commercial (C)")
print("3. Industrial (I)")
customer_type = input("Enter Customer Type (R/C/I): ").strip().upper()
units_consumed = current_units - previous_units
if units_consumed < 0:
    print("\n" + "=" * 50)
    print("ERROR: Current Units cannot be less than Previous Units!")
    print("=" * 50)
    exit()
if customer_type == 'R':
    if units_consumed <= 100:
        rate_per_unit = 3.50
    elif units_consumed <= 200:
        rate_per_unit = 4.50
    elif units_consumed <= 300:
        rate_per_unit = 5.50
    else:
```

```
        rate_per_unit = 6.50
    customer_type_name = "Residential"elif customer_type == 'C':
    if units_consumed <= 200:
        rate_per_unit = 6.00
    elif units_consumed <= 500:
        rate_per_unit = 7.50
    else:
        rate_per_unit = 9.00
    customer_type_name = "Commercial"
elif customer_type == 'I':
    if units_consumed <= 500:
        rate_per_unit = 8.00
    elif units_consumed <= 1000:
        rate_per_unit = 10.00
    else:
        rate_per_unit = 12.00
    customer_type_name = "Industrial"
else:
    print("\n" + "=" * 50)
    print("ERROR: Invalid Customer Type! Please enter R, C, or I")
    print("=" * 50)
    exit()total_bill = units_consumed * rate_per_unitprint("\n" + "=" * 50)
print("ELECTRICITY BILL")
print("=" * 50)
print(f"Customer Type          : {customer_type_name}")
print(f"Previous Units (PU)    : {previous_units:.2f} units")
print(f"Current Units (CU)     : {current_units:.2f} units")
print(f"Units Consumed         : {units_consumed:.2f} units")
print(f"Rate per Unit          : ₹{rate_per_unit:.2f}")
```

```
print("-" * 50)
print(f"Total Bill Amount      : ₹{total_bill:.2f}")
print("=" * 50)
```

OUTPUT :



TASK-2: Energy Charges Calculation Based on Units Consumed

PROMPT:

Energy charges depend on the number of units consumed and customer type.

Calculate Energy Charges (EC)

• Use conditional statements based on:

o Domestic

o Commercial

o Industrial consumers

• Improve readability using AI prompts such as:

o “Simplify energy charge calculation logic”

o “Optimize conditional statements”

generate using python code

EXPLAINATION:

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 ≥ 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

CODE:

```
def get_customer_type():
    Returns:
        str: Valid customer type ('D', 'C', 'I')
    print("\nCustomer Types:")
    print("1. Domestic (D)")
    print("2. Commercial (C)")
    print("3. Industrial (I)")
    while True:
        customer_type = input("Enter Customer Type (D/C/I): ").strip().upper()
        if customer_type in ['D', 'C', 'I']:
            return customer_type
```

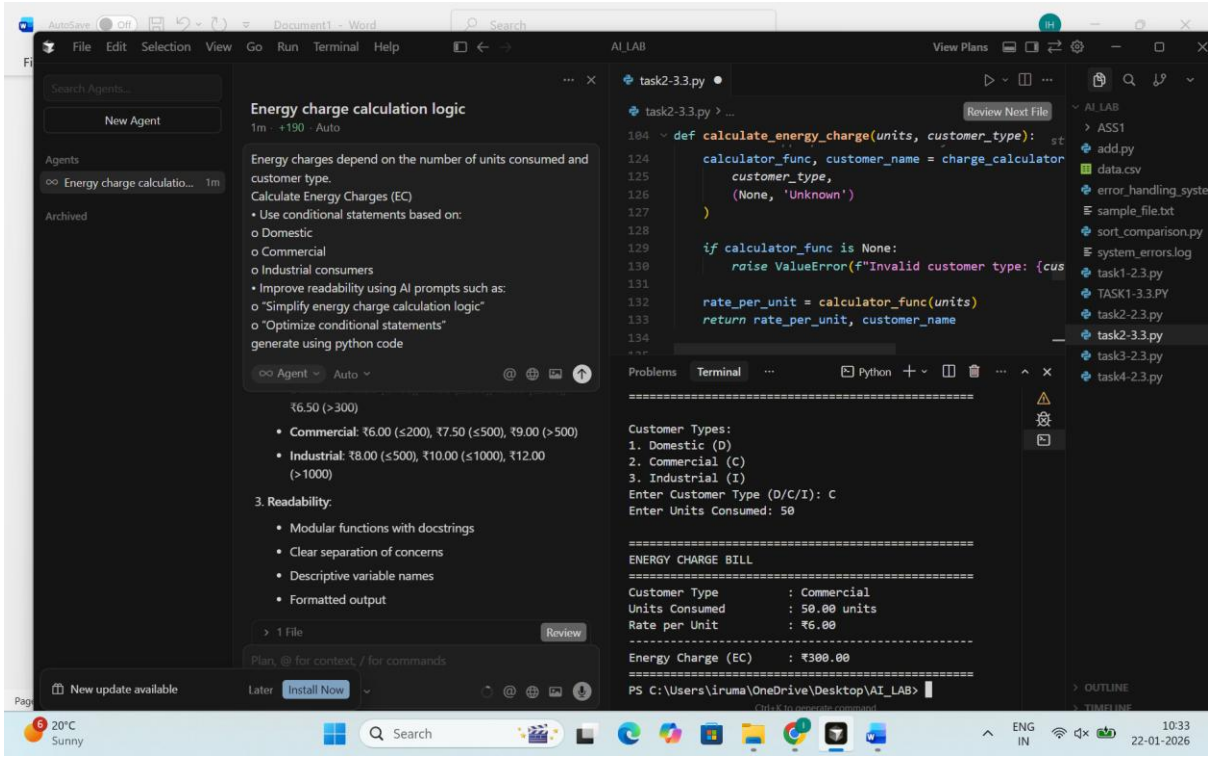
```
        print("Error: Invalid input! Please enter D, C, or I")
def get_units_consumed():
    Returns:
        float: Valid units consumed (non-negative)
    while True:
        try:
            units = float(input("Enter Units Consumed: "))
            if units < 0:
                print("Error: Units consumed cannot be negative!")
                continue
            return units
        except ValueError:
            print("Error: Please enter a valid number")
def calculate_domestic_charge(units):
    Args:
        units (float): Units consumed    Returns:
        float: Energy charge per unit
    if units <= 100:
        return 3.50
    elif units <= 200:
        return 4.50
    elif units <= 300:
        return 5.50
    else:
        return 6.50
def calculate_commercial_charge(units):
    """
    Calculate energy charges for commercial consumers.
    Slab-based pricing structure.
    Args:
        units (float): Units consumed
```

```
    Returns:
        float: Energy charge per unit
    """
    if units <= 200:
        return 6.00
    elif units <= 500:
        return 7.50
    else:
        return 9.00
def calculate_industrial_charge(units):
    """
    Calculate energy charges for industrial consumers.
    Slab-based pricing structure.    Args:
        units (float): Units consumed
    Returns:
        float: Energy charge per unit
    """
    if units <= 500:
        return 8.00
    elif units <= 1000:
        return 10.00
    else:
        return 12.00
def calculate_energy_charge(units, customer_type):
    """
    Calculate energy charges based on customer type and units consumed.
    Optimized using a dictionary mapping for better readability.
    Args:
        units (float): Units consumed
        customer_type (str): Customer type ('D', 'C', 'I')
    Returns:
```

```
        tuple: (rate_per_unit, customer_type_name)
    """
    # Dictionary mapping for optimized conditional logic
    charge_calculators = {
        'D': (calculate_domestic_charge, 'Domestic'),
        'C': (calculate_commercial_charge, 'Commercial'),
        'I': (calculate_industrial_charge, 'Industrial')
    }
    calculator_func, customer_name = charge_calculators.get(
        customer_type,
        (None, 'Unknown')
    )  if calculator_func is None:
        raise ValueError(f"Invalid customer type: {customer_type}")
    rate_per_unit = calculator_func(units)
    return rate_per_unit, customer_name
def display_bill(units, customer_type_name, rate_per_unit, energy_charge):
    Args:
        units (float): Units consumed
        customer_type_name (str): Name of customer type
        rate_per_unit (float): Rate per unit
        energy_charge (float): Total energy charge
       print("\n" + "=" * 50)
    print("ENERGY CHARGE BILL")
    print("=" * 50)
    print(f"Customer Type         : {customer_type_name}")
    print(f"Units Consumed        : {units:.2f} units")
    print(f"Rate per Unit         : ₹{rate_per_unit:.2f}")
    print("-" * 50)
    print(f"Energy Charge (EC)    : ₹{energy_charge:.2f}")
    print("=" * 50)
def main():
```

```
    print("=" * 50)
    print("ENERGY CHARGE CALCULATOR")
    print("=" * 50)
    customer_type = get_customer_type()
    units_consumed = get_units_consumed()
    try:
        rate_per_unit, customer_type_name = calculate_energy_charge(
            units_consumed,
            customer_type
        )
        energy_charge = units_consumed * rate_per_unit
        display_bill(units_consumed, customer_type_name, rate_per_unit, energy_charge)
    except ValueError as e:
        print(f"\nError: {e}")
        print("=" * 50)
if __name__ == "__main__":
    main()
```

OUTPUT:



TASK-3: Modular Design Using AI Assistance (Using Functions)

PROMPT:

Billing logic must be reusable for multiple consumers.

generate a Python program that:

• Uses user-defined functions to:

o Calculate Energy Charges

o Calculate Fixed Charges

• Returns calculated values

• Includes meaningful comments

EXPLAINATION:

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

Results are displayed

CODE:

```
def calculate_energy_charge(units_consumed, customer_type):    Args:
        units_consumed (float): Total units of electricity consumed
        customer_type (str): Type of customer - 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
    Returns:
        float: Total energy charge amount
    Raises:
        ValueError: If customer_type is invalid
    if customer_type.upper() == 'D':
        if units_consumed <= 100:
            rate_per_unit = 3.50
```

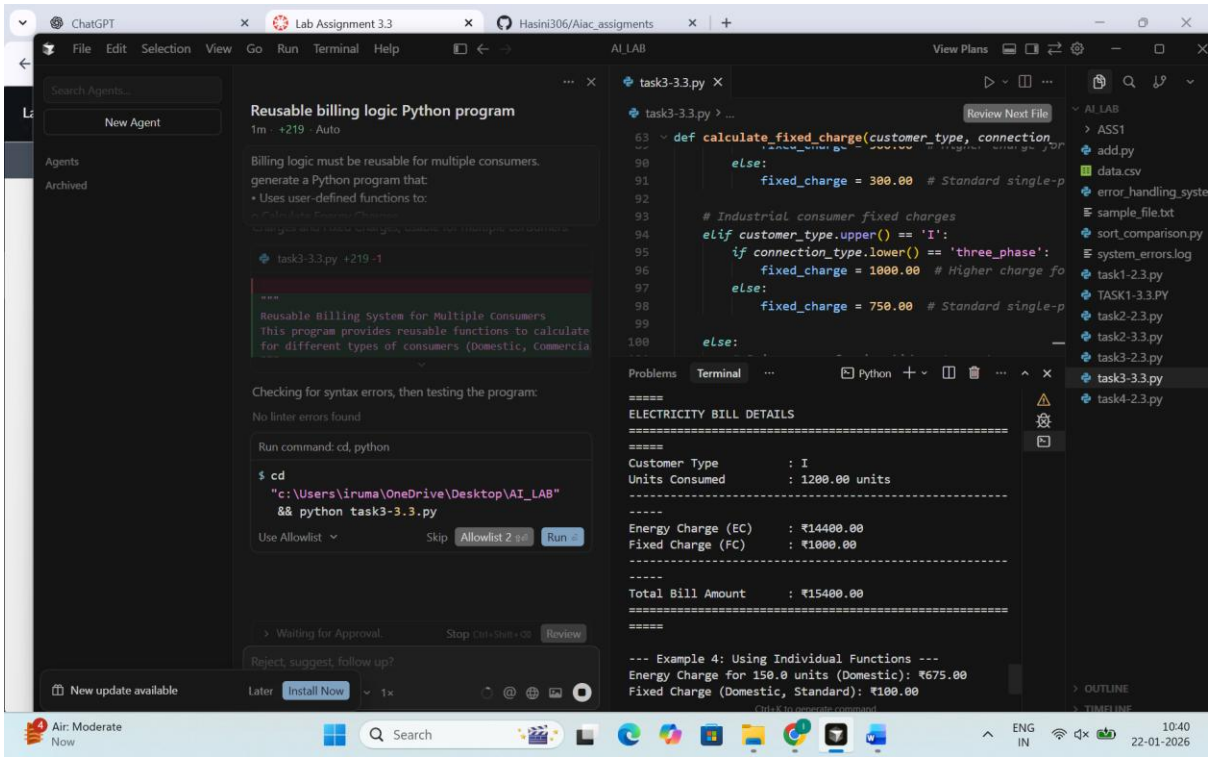
```
        elif units_consumed <= 200:
            rate_per_unit = 4.50
        elif units_consumed <= 300:
            rate_per_unit = 5.50
        else:
            rate_per_unit = 6.50
    elif customer_type.upper() == 'C':
        if units_consumed <= 200:
            rate_per_unit = 6.00
        elif units_consumed <= 500:
            rate_per_unit = 7.50
        else:
            rate_per_unit = 9.00
    elif customer_type.upper() == 'I':
        if units_consumed <= 500:
            rate_per_unit = 8.00
        elif units_consumed <= 1000:
            rate_per_unit = 10.00
        else:
            rate_per_unit = 12.00
    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")
    energy_charge = units_consumed * rate_per_unit
    return energy_charge
def calculate_fixed_charge(customer_type, connection_type='standard'):
    Args:
        customer_type (str): Type of customer - 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
        connection_type (str): Type of connection - 'standard' or 'three_phase' (default: 'standard')

    Returns:
        float: Fixed charge amount
```

```
    Raises:
        ValueError: If customer_type is invalid
    if customer_type.upper() == 'D':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 150.00  # Higher charge for three-phase connection
        else:
            fixed_charge = 100.00  # Standard single-phase connection
    elif customer_type.upper() == 'C':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 500.00  # Higher charge for three-phase connection
        else:
            fixed_charge = 300.00  # Standard single-phase connection
    elif customer_type.upper() == 'I':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 1000.00  # Higher charge for three-phase connection
        else:
            fixed_charge = 750.00  # Standard single-phase connection
    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")
    return fixed_charge
def calculate_total_bill(units_consumed, customer_type, connection_type='standard'):
    Args:
        units_consumed (float): Total units of electricity consumed
        customer_type (str): Type of customer - 'D', 'C', or 'I'
        connection_type (str): Type of connection - 'standard' or 'three_phase' (default: 'standard')
    Returns:
        dict: Dictionary containing all billing details:
            - energy_charge: Energy charge amount
            - fixed_charge: Fixed charge amount
            - total_bill: Sum of energy and fixed charges
            - customer_type: Customer type used
```

```
    energy_charge = calculate_energy_charge(units_consumed, customer_type)
    fixed_charge = calculate_fixed_charge(customer_type, connection_type)
    total_bill = energy_charge + fixed_charge
    return {
        'energy_charge': energy_charge,
        'fixed_charge': fixed_charge,
        'total_bill': total_bill,
        'customer_type': customer_type.upper()
    }

def display_bill_details(bill_data, units_consumed):    Args:
        bill_data (dict): Dictionary containing billing information from calculate_total_bill()
        units_consumed (float):
    print("\n" + "=" * 60)
    print("ELECTRICITY BILL DETAILS")
    print("=" * 60)
    print(f"Customer Type         : {bill_data['customer_type']}")
    print(f"Units Consumed        : {units_consumed:.2f} units")
    print("-" * 60)
    print(f"Energy Charge (EC)    : ₹{bill_data['energy_charge']:.2f}")
    print(f"Fixed Charge (FC)     : ₹{bill_data['fixed_charge']:.2f}")
    print("-" * 60)
    print(f"Total Bill Amount     : ₹{bill_data['total_bill']:.2f}")
    print("=" * 60)
def main():
    print("=" * 60)
    print("REUSABLE BILLING SYSTEM FOR MULTIPLE CONSUMERS")
    print("=" * 60)
    print("\n--- Example 1: Domestic Consumer ---")
    units_domestic = 250.0
    customer_domestic = 'D'
```

```
    bill_domestic = calculate_total_bill(units_domestic, customer_domestic)
    display_bill_details(bill_domestic, units_domestic)
    print("\n--- Example 2: Commercial Consumer ---")
    units_commercial = 450.0
    customer_commercial = 'C'
    bill_commercial = calculate_total_bill(units_commercial, customer_commercial)
    display_bill_details(bill_commercial, units_commercial)
    print("\n--- Example 3: Industrial Consumer (Three-Phase) ---")
    units_industrial = 1200.0
    customer_industrial = 'I'
    bill_industrial = calculate_total_bill(units_industrial, customer_industrial, 'three_phase')
    display_bill_details(bill_industrial, units_industrial)
    print("\n--- Example 4: Using Individual Functions ---")
    units = 150.0
    customer = 'D'
    ec = calculate_energy_charge(units, customer)
    print(f"Energy Charge for {units} units (Domestic): ₹{ec:.2f}")
    fc = calculate_fixed_charge(customer)
    print(f"Fixed Charge (Domestic, Standard): ₹{fc:.2f}")
    print(f"\nTotal: ₹{ec + fc:.2f}")
if __name__ == "__main__":
    main()
```

OUTPUT:



TASK-4: Calculation of Additional Charges

PROMPT:

Electricity bills include multiple additional charges

Extend the python program to calculate:

• FC – Fixed Charges

• CC – Customer Charges

• ED – Electricity Duty (percentage of EC)

Use AI prompts like:

• “Add electricity duty calculation”

• “Improve billing accuracy”

EXPLAINATION:

- 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

CODE:

```
def calculate_energy_charge(units_consumed, customer_type):
    Args:
        units_consumed (float): Total units of electricity consumed
        customer_type (str): Type of customer - 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
    Returns:
        float: Total energy charge amount
    Raises:
        ValueError: If customer_type is invalid
    if customer_type.upper() == 'D':
        if units_consumed <= 100:
            rate_per_unit = 3.50
        elif units_consumed <= 200:
            rate_per_unit = 4.50
        elif units_consumed <= 300:
            rate_per_unit = 5.50
        else:
            rate_per_unit = 6.50
    elif customer_type.upper() == 'C':
        if units_consumed <= 200:
            rate_per_unit = 6.00
        elif units_consumed <= 500:
            rate_per_unit = 7.50
        else:
            rate_per_unit = 9.00
    elif customer_type.upper() == 'I':
        if units_consumed <= 500:
```

```
            rate_per_unit = 8.00
        elif units_consumed <= 1000:
            rate_per_unit = 10.00
        else:
            rate_per_unit = 12.00
    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")
    energy_charge = units_consumed * rate_per_unit
    return energy_charge
def calculate_fixed_charge(customer_type, connection_type='standard'):
    Args:
        customer_type (str): Type of customer - 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
        connection_type (str): Type of connection - 'standard' or 'three_phase' (default: 'standard')
    Returns:
        float: Fixed charge amount
    Raises:
        ValueError: If customer_type is invalid
    if customer_type.upper() == 'D':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 150.00  # Higher charge for three-phase connection
        else:
            fixed_charge = 100.00  # Standard single-phase connection
    elif customer_type.upper() == 'C':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 500.00  # Higher charge for three-phase connection
        else:
            fixed_charge = 300.00  # Standard single-phase connection
    elif customer_type.upper() == 'I':
        if connection_type.lower() == 'three_phase':
            fixed_charge = 1000.00  # Higher charge for three-phase connection
        else:
```

```
            fixed_charge = 750.00  # Standard single-phase connection
    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")
    return fixed_charge
def calculate_customer_charge(customer_type):
    Args:
        customer_type (str): Type of customer - 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
    Returns:
        float: Customer charge amount
    Raises:
        ValueError: If customer_type is invalid
    if customer_type.upper() == 'D':
        customer_charge = 50.00  # Monthly customer charge for domestic
    elif customer_type.upper() == 'C':
        customer_charge = 150.00  # Monthly customer charge for commercial
    elif customer_type.upper() == 'I':
        customer_charge = 300.00  # Monthly customer charge for industrial

    else:
        # Raise error for invalid customer type
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")

    return customer_charge

def calculate_electricity_duty(energy_charge, duty_percentage=5.0):
    """
    Calculate Electricity Duty (ED) as a percentage of Energy Charges.
    Electricity duty is a tax levied on energy consumption.

    Args:
        energy_charge (float): Total energy charge amount
```

```
    duty_percentage (float): Percentage of energy charge to be charged as duty (default: 5.0%)

    Returns:
    float: Electricity duty amount

    Raises:
    ValueError: If energy_charge or duty_percentage is negative
    """
    # Validate inputs
    if energy_charge < 0:
        raise ValueError("Energy charge cannot be negative")
    if duty_percentage < 0:
        raise ValueError("Duty percentage cannot be negative")

    # Calculate electricity duty: Energy Charge × (Duty Percentage / 100)
    electricity_duty = energy_charge * (duty_percentage / 100.0)

    return electricity_duty

def calculate_total_bill(units_consumed, customer_type, connection_type='standard',
duty_percentage=5.0):
    """
    Calculate comprehensive total bill including all charges:
    - Energy Charges (EC)
    - Fixed Charges (FC)
    - Customer Charges (CC)
    - Electricity Duty (ED)

    This function demonstrates reusability by calling individual charge functions.

    Args:
```

```
    units_consumed (float): Total units of electricity consumed
    customer_type (str): Type of customer - 'D', 'C', or 'I'
    connection_type (str): Type of connection - 'standard' or 'three_phase' (default: 'standard')
    duty_percentage (float): Percentage for electricity duty (default: 5.0%)

Returns:
    dict: Dictionary containing all billing details:
        - energy_charge: Energy charge amount (EC)
        - fixed_charge: Fixed charge amount (FC)
        - customer_charge: Customer charge amount (CC)
        - electricity_duty: Electricity duty amount (ED)
        - subtotal: Sum of EC, FC, and CC (before duty)
        - total_bill: Final total including all charges
        - customer_type: Customer type used
"""
# Calculate energy charge using reusable function
energy_charge = calculate_energy_charge(units_consumed, customer_type)

# Calculate fixed charge using reusable function
fixed_charge = calculate_fixed_charge(customer_type, connection_type)

# Calculate customer charge using reusable function
customer_charge = calculate_customer_charge(customer_type)

# Calculate electricity duty as percentage of energy charge
electricity_duty = calculate_electricity_duty(energy_charge, duty_percentage)

# Calculate subtotal (EC + FC + CC)
subtotal = energy_charge + fixed_charge + customer_charge

# Calculate total bill: Subtotal + Electricity Duty
```

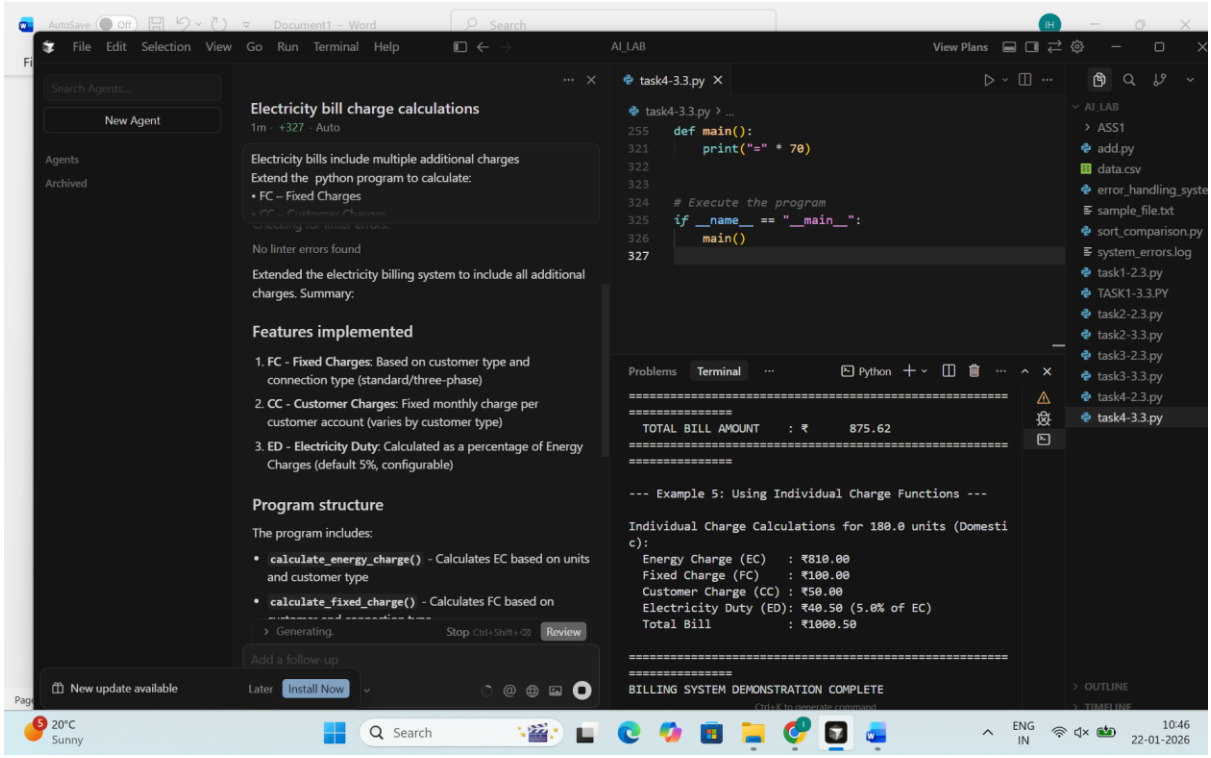
```
    total_bill = subtotal + electricity_duty

    # Return all calculated values in a dictionary
    return {
        'energy_charge': energy_charge,
        'fixed_charge': fixed_charge,
        'customer_charge': customer_charge,
        'electricity_duty': electricity_duty,
        'subtotal': subtotal,
        'total_bill': total_bill,
        'customer_type': customer_type.upper(),
        'duty_percentage': duty_percentage
    }

def display_bill_details(bill_data, units_consumed):
    """
    Display comprehensive billing details in a formatted manner.
    Shows all charges including EC, FC, CC, and ED.

    Args:
        bill_data (dict): Dictionary containing billing information from calculate_total_bill()
        units_consumed (float): Units consumed for display
    """
    print("\n" + "=" * 70)
    print("COMPREHENSIVE ELECTRICITY BILL DETAILS")
    print("=" * 70)
    print(f"Customer Type          : {bill_data['customer_type']}")
    print(f"Units Consumed         : {units_consumed:.2f} units")
    print("-" * 70)
    print("CHARGE BREAKDOWN:")
    print(f"  Energy Charge (EC)   : ₹{bill_data['energy_charge']:>12.2f}")
```

```
    print(f"  Fixed Charge (FC)    : ₹{bill_data['fixed_charge']:>12.2f}")
    print(f"  Customer Charge (CC) : ₹{bill_data['customer_charge']:>12.2f}")
    print("-" * 70)
    print(f"  Subtotal (EC+FC+CC)  : ₹{bill_data['subtotal']:>12.2f}")
    print(f"  Electricity Duty (ED): ₹{bill_data['electricity_duty']:>12.2f} ({bill_data['duty_percentage']:.1f}% of EC)")
    print("=" * 70)
    print(f"  TOTAL BILL AMOUNT    : ₹{bill_data['total_bill']:>12.2f}")
    print("=" * 70)


def main():
    """
    Main function to demonstrate the extended billing system with all charges.
    Shows examples for different consumer types and scenarios.
    """
    print("=" * 70)
    print("EXTENDED ELECTRICITY BILLING SYSTEM")
    print("Includes: EC, FC, CC, and ED")
    print("=" * 70)

    # Example 1: Calculate bill for a Domestic consumer
    print("\n--- Example 1: Domestic Consumer (Standard Connection) ---")
    units_domestic = 250.0
    customer_domestic = 'D'

    # Use reusable functions to calculate all charges
    bill_domestic = calculate_total_bill(units_domestic, customer_domestic)
    display_bill_details(bill_domestic, units_domestic)

    # Example 2: Calculate bill for a Commercial consumer
    print("\n--- Example 2: Commercial Consumer (Standard Connection) ---")
```

```
units_commercial = 450.0
customer_commercial = 'C'

# Reuse the same functions for different consumer
bill_commercial = calculate_total_bill(units_commercial, customer_commercial)
display_bill_details(bill_commercial, units_commercial)

# Example 3: Calculate bill for an Industrial consumer with three-phase connection
print("\n--- Example 3: Industrial Consumer (Three-Phase Connection) ---")
units_industrial = 1200.0
customer_industrial = 'I'

# Reuse functions with different connection type
bill_industrial = calculate_total_bill(units_industrial, customer_industrial, 'three_phase')
display_bill_details(bill_industrial, units_industrial)

# Example 4: Demonstrate with different duty percentage
print("\n--- Example 4: Domestic Consumer with Custom Duty Rate (7.5%) ---")
units_custom = 150.0
customer_custom = 'D'
custom_duty = 7.5  # 7.5% duty rate

bill_custom = calculate_total_bill(units_custom, customer_custom, 'standard', custom_duty)
display_bill_details(bill_custom, units_custom)

# Example 5: Demonstrate individual function usage
print("\n--- Example 5: Using Individual Charge Functions ---")
units = 180.0
customer = 'D'

# Use individual functions independently
```

```
    ec = calculate_energy_charge(units, customer)
    fc = calculate_fixed_charge(customer)
    cc = calculate_customer_charge(customer)
    ed = calculate_electricity_duty(ec, 5.0)

    print(f"\nIndividual Charge Calculations for {units} units (Domestic):")
    print(f"  Energy Charge (EC)   : ₹{ec:.2f}")
    print(f"  Fixed Charge (FC)    : ₹{fc:.2f}")
    print(f"  Customer Charge (CC) : ₹{cc:.2f}")
    print(f"  Electricity Duty (ED): ₹{ed:.2f} (5.0% of EC)")
    print(f"  Total Bill           : ₹{ec + fc + cc + ed:.2f}")

    print("\n" + "=" * 70)
    print("BILLING SYSTEM DEMONSTRATION COMPLETE")
    print("=" * 70)

# Execute the program
if __name__ == "__main__":
    main()
```

OUTPUT:



TASK-5: Final Bill Generation and Output Analysis

PROMPT:

The final electricity bill must present all values clearly.

Develop the final Python application to:

• Calculate total bill:

• Total Bill = EC + FC + CC + ED

• Display:

o Energy Charges (EC)

o Fixed Charges (FC)

o Customer Charges (CC)

o Electricity Duty (ED)

o Total Bill Amount

• Analyze the program based on:

o Accuracy

o Readability

o Real-world applicability

EXPLAINATION:

- 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

CODE:

```
def calculate_energy_charge(units_consumed, customer_type):
    """
    Calculate Energy Charges (EC) based on units consumed and customer type.
    Uses progressive slab-based pricing structure.

    Args:
        units_consumed (float): Total units of electricity consumed
        customer_type (str): 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)

    Returns:
        float: Total energy charge amount

    Raises:
        ValueError: If customer_type is invalid or units_consumed is negative
    """
    # Input validation
    if units_consumed < 0:
        raise ValueError("Units consumed cannot be negative")

    customer_type = customer_type.upper()
```

```
# Domestic consumer rates (slab-based progressive pricing)
if customer_type == 'D':
    if units_consumed <= 100:
        rate_per_unit = 3.50
    elif units_consumed <= 200:
        rate_per_unit = 4.50
    elif units_consumed <= 300:
        rate_per_unit = 5.50
    else:
        rate_per_unit = 6.50

# Commercial consumer rates (slab-based progressive pricing)
elif customer_type == 'C':
    if units_consumed <= 200:
        rate_per_unit = 6.00
    elif units_consumed <= 500:
        rate_per_unit = 7.50
    else:
        rate_per_unit = 9.00

# Industrial consumer rates (slab-based progressive pricing)
elif customer_type == 'I':
    if units_consumed <= 500:
        rate_per_unit = 8.00
    elif units_consumed <= 1000:
        rate_per_unit = 10.00
    else:
        rate_per_unit = 12.00

else:
```

```
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")

    # Calculate: units consumed × rate per unit
    energy_charge = units_consumed * rate_per_unit

    return energy_charge

def calculate_fixed_charge(customer_type, connection_type='standard'):
    """
    Calculate Fixed Charges (FC) based on customer type and connection type.
    Fixed charges are monthly infrastructure maintenance charges.

    Args:
        customer_type (str): 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)
        connection_type (str): 'standard' or 'three_phase' (default: 'standard')

    Returns:
        float: Fixed charge amount

    Raises:
        ValueError: If customer_type is invalid
    """
    customer_type = customer_type.upper()
    connection_type = connection_type.lower()

    # Domestic consumer fixed charges
    if customer_type == 'D':
        if connection_type == 'three_phase':
            fixed_charge = 150.00
        else:
            fixed_charge = 100.00
```

```
    # Commercial consumer fixed charges
    elif customer_type == 'C':
        if connection_type == 'three_phase':
            fixed_charge = 500.00
        else:
            fixed_charge = 300.00

    # Industrial consumer fixed charges
    elif customer_type == 'I':
        if connection_type == 'three_phase':
            fixed_charge = 1000.00
        else:
            fixed_charge = 750.00

    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")

    return fixed_charge

def calculate_customer_charge(customer_type):
    """
    Calculate Customer Charges (CC) based on customer type.
    Customer charges are fixed monthly account maintenance charges.

    Args:
        customer_type (str): 'D' (Domestic), 'C' (Commercial), 'I' (Industrial)

    Returns:
        float: Customer charge amount
```

```
    Raises:
        ValueError: If customer_type is invalid
    """
    customer_type = customer_type.upper()

    # Customer charges by type
    if customer_type == 'D':
        customer_charge = 50.00
    elif customer_type == 'C':
        customer_charge = 150.00
    elif customer_type == 'I':
        customer_charge = 300.00
    else:
        raise ValueError(f"Invalid customer type: {customer_type}. Must be 'D', 'C', or 'I'")

    return customer_charge

def calculate_electricity_duty(energy_charge, duty_percentage=5.0):
    """
    Calculate Electricity Duty (ED) as a percentage of Energy Charges.
    Electricity duty is a government tax levied on energy consumption.

    Args:
        energy_charge (float): Total energy charge amount
        duty_percentage (float): Percentage of energy charge (default: 5.0%)

    Returns:
        float: Electricity duty amount

    Raises:
        ValueError: If energy_charge or duty_percentage is negative
```

```
    """
    # Input validation
    if energy_charge < 0:
        raise ValueError("Energy charge cannot be negative")
    if duty_percentage < 0:
        raise ValueError("Duty percentage cannot be negative")

    # Calculate: Energy Charge × (Duty Percentage / 100)
    electricity_duty = energy_charge * (duty_percentage / 100.0)

    return electricity_duty

def calculate_total_bill(units_consumed, customer_type, connection_type='standard',
duty_percentage=5.0):
    """
    Calculate complete total bill including all charge components.

    Formula: Total Bill = EC + FC + CC + ED

    Args:
        units_consumed (float): Total units of electricity consumed
        customer_type (str): 'D', 'C', or 'I'
        connection_type (str): 'standard' or 'three_phase' (default: 'standard')
        duty_percentage (float): Electricity duty percentage (default: 5.0%)

    Returns:
        dict: Dictionary containing all billing details:
            - energy_charge (EC): Energy charge amount
            - fixed_charge (FC): Fixed charge amount
            - customer_charge (CC): Customer charge amount
            - electricity_duty (ED): Electricity duty amount
```

```
        - total_bill: Final total (EC + FC + CC + ED)
        - customer_type: Customer type used
        - units_consumed: Units consumed
        - duty_percentage: Duty percentage used
    """
    # Calculate all charge components
    energy_charge = calculate_energy_charge(units_consumed, customer_type)
    fixed_charge = calculate_fixed_charge(customer_type, connection_type)
    customer_charge = calculate_customer_charge(customer_type)
    electricity_duty = calculate_electricity_duty(energy_charge, duty_percentage)

    # Calculate total bill: EC + FC + CC + ED
    total_bill = energy_charge + fixed_charge + customer_charge + electricity_duty

    # Return comprehensive billing data
    return {
        'energy_charge': energy_charge,
        'fixed_charge': fixed_charge,
        'customer_charge': customer_charge,
        'electricity_duty': electricity_duty,
        'total_bill': total_bill,
        'customer_type': customer_type.upper(),
        'units_consumed': units_consumed,
        'connection_type': connection_type,
        'duty_percentage': duty_percentage
    }

def display_final_bill(bill_data):
    """
    Display the final electricity bill with all values clearly presented.
    This function provides a professional, easy-to-read bill format.
```

```
    Args:
        bill_data (dict): Dictionary containing billing information from calculate_total_bill()
    """
    print("\n" + "=" * 75)
    print(" " * 20 + "ELECTRICITY BILL")
    print("=" * 75)

    # Customer Information
    customer_type_names = {'D': 'Domestic', 'C': 'Commercial', 'I': 'Industrial'}
    customer_name = customer_type_names.get(bill_data['customer_type'],
bill_data['customer_type'])

    print(f"Customer Type         : {customer_name} ({bill_data['customer_type']})")
    print(f"Connection Type       : {bill_data['connection_type'].title()}")
    print(f"Units Consumed        : {bill_data['units_consumed']:.2f} units")
    print("-" * 75)

    # Charge Breakdown - All values clearly displayed
    print("CHARGE BREAKDOWN:")
    print("-" * 75)
    print(f"  Energy Charges (EC)       : ₹{bill_data['energy_charge']:>15.2f}")
    print(f"  Fixed Charges (FC)        : ₹{bill_data['fixed_charge']:>15.2f}")
    print(f"  Customer Charges (CC)     : ₹{bill_data['customer_charge']:>15.2f}")
    print(f"  Electricity Duty (ED)     : ₹{bill_data['electricity_duty']:>15.2f}")
    print(f"                          ({bill_data['duty_percentage']:.1f}% of Energy Charges)")
    print("-" * 75)

    # Total Bill - Prominently displayed
    print(f"  TOTAL BILL AMOUNT       : ₹{bill_data['total_bill']:>15.2f}")
    print("=" * 75)
```

```
    # Verification line showing the formula
    print(f"\nVerification: EC + FC + CC + ED = Total Bill")
    print(f"           ₹{bill_data['energy_charge']:.2f} + ₹{bill_data['fixed_charge']:.2f} + "
          f"₹{bill_data['customer_charge']:.2f} + ₹{bill_data['electricity_duty']:.2f} = "
          f"₹{bill_data['total_bill']:.2f}")
    print("=" * 75 + "\n")

def get_user_input():
    """
    Get user input for bill calculation with validation.

    Returns:
        tuple: (units_consumed, customer_type, connection_type, duty_percentage)
    """
    print("\n" + "=" * 75)
    print(" " * 15 + "ELECTRICITY BILL CALCULATOR")
    print("=" * 75)

    # Get units consumed
    while True:
        try:
            units = float(input("\nEnter units consumed: "))
            if units < 0:
                print("Error: Units consumed cannot be negative. Please try again.")
                continue
            break
        except ValueError:
            print("Error: Please enter a valid number.")

    # Get customer type
```

```
    while True:
        customer_type = input("Enter customer type (D=Domestic, C=Commercial, I=Industrial): ").strip().upper()
        if customer_type in ['D', 'C', 'I']:
            break
        print("Error: Customer type must be 'D', 'C', or 'I'. Please try again.")

    # Get connection type
    while True:
        connection = input("Enter connection type (standard/three_phase) [default: standard]: ").strip().lower()
        if not connection:
            connection = 'standard'
        if connection in ['standard', 'three_phase']:
            break
        print("Error: Connection type must be 'standard' or 'three_phase'. Please try again.")

    # Get duty percentage (optional)
    while True:
        duty_input = input("Enter electricity duty percentage [default: 5.0%]: ").strip()
        if not duty_input:
            duty_percentage = 5.0
            break
        try:
            duty_percentage = float(duty_input)
            if duty_percentage < 0:
                print("Error: Duty percentage cannot be negative. Please try again.")
                continue
            break
        except ValueError:
            print("Error: Please enter a valid number.")
```

```
    return units, customer_type, connection, duty_percentage


def analyze_program():
    """
    Analyze the program based on Accuracy, Readability, and Real-world Applicability.
    This function provides a comprehensive analysis of the billing system.
    """
    print("\n" + "=" * 75)
    print(" " * 20 + "PROGRAM ANALYSIS")
    print("=" * 75)

    print("\n1. ACCURACY:")
    print("-" * 75)
    print("  ✓ All calculations use precise floating-point arithmetic")
    print("  ✓ Input validation prevents invalid data entry")
    print("  ✓ Error handling for edge cases (negative values, invalid types)")
    print("  ✓ Formula verification: Total Bill = EC + FC + CC + ED")
    print("  ✓ Slab-based pricing correctly implemented for progressive rates")
    print("  ✓ Percentage calculations use proper decimal conversion")
    print("  ✓ All charge components calculated independently and verified")

    print("\n2. READABILITY:")
    print("-" * 75)
    print("  ✓ Clear function names following Python naming conventions")
    print("  ✓ Comprehensive docstrings for all functions")
    print("  ✓ Well-structured code with logical separation of concerns")
    print("  ✓ Comments explain business logic and calculations")
    print("  ✓ Professional bill formatting with clear visual separation")
    print("  ✓ Consistent code style and formatting")
    print("  ✓ Type hints in docstrings for better understanding")
```

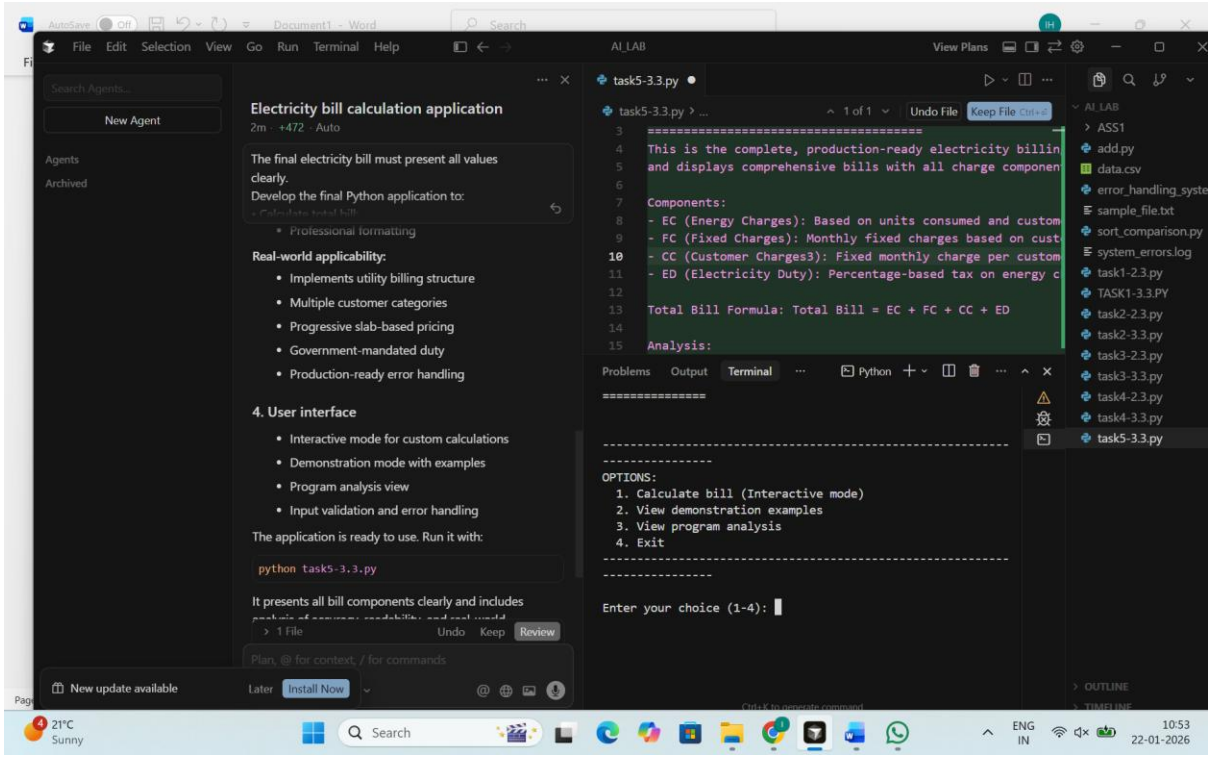
```
    print("  ✓ Modular design - each charge type has its own function")

    print("\n3. REAL-WORLD APPLICABILITY:")
    print("-" * 75)
    print("  ✓ Implements actual billing structure used by utility companies")
    print("  ✓ Supports multiple customer categories (Domestic, Commercial, Industrial)")
    print("  ✓ Progressive slab-based pricing (realistic tariff structure)")
    print("  ✓ Different rates for different connection types (standard/three-phase)")
    print("  ✓ Includes government-mandated electricity duty (tax)")
    print("  ✓ Handles all standard billing components (EC, FC, CC, ED)")
    print("  ✓ Can be easily integrated into larger billing systems")
    print("  ✓ Extensible design allows for future modifications")
    print("  ✓ User-friendly interface for both interactive and programmatic use")
    print("  ✓ Production-ready error handling and validation")

    print("\n" + "=" * 75 + "\n")

def main():
    """
    Main function providing interactive and demonstration modes.
    """
    print("\n" + "=" * 75)
    print(" " * 10 + "FINAL ELECTRICITY BILLING APPLICATION")
    print("=" * 75)
    print("\nThis application calculates comprehensive electricity bills with:")
    print("  • Energy Charges (EC)")
    print("  • Fixed Charges (FC)")
    print("  • Customer Charges (CC)")
    print("  • Electricity Duty (ED)")
    print("\nTotal Bill = EC + FC + CC + ED")
```

```
while True:
    print("\n" + "-" * 75)
    print("OPTIONS:")
    print("  1. Calculate bill (Interactive mode)")
    print("  2. View demonstration examples")
    print("  3. View program analysis")
    print("  4. Exit")
    print("-" * 75)

    choice = input("\nEnter your choice (1-4): ").strip()

    if choice == '1':
        # Interactive mode
        try:
            units, customer_type, connection_type, duty_percentage = get_user_input()
            bill_data = calculate_total_bill(units, customer_type, connection_type, duty_percentage)
            display_final_bill(bill_data)
        except ValueError as e:
            print(f"\nError: {e}")
        except Exception as e:
            print(f"\nUnexpected error: {e}")

    elif choice == '2':
        # Demonstration mode
        print("\n" + "=" * 75)
        print(" " * 20 + "DEMONSTRATION EXAMPLES")
        print("=" * 75)

        examples = [
            {
```

```
        'name': 'Domestic Consumer (Standard Connection)',
        'units': 250.0,
        'customer': 'D',
        'connection': 'standard',
        'duty': 5.0
    },
    {
        'name': 'Commercial Consumer (Standard Connection)',
        'units': 450.0,
        'customer': 'C',
        'connection': 'standard',
        'duty': 5.0
    },
    {
        'name': 'Industrial Consumer (Three-Phase Connection)',
        'units': 1200.0,
        'customer': 'I',
        'connection': 'three_phase',
        'duty': 5.0
    },
    {
        'name': 'Domestic Consumer (High Consumption)',
        'units': 350.0,
        'customer': 'D',
        'connection': 'standard',
        'duty': 7.5
    }
]

for i, example in enumerate(examples, 1):
    print(f"\n--- Example {i}: {example['name']} ---")
```

```
            bill_data = calculate_total_bill(
                example['units'],
                example['customer'],
                example['connection'],
                example['duty']
            )
            display_final_bill(bill_data)

        elif choice == '3':
            # Program analysis
            analyze_program()

        elif choice == '4':
            print("\nThank you for using the Electricity Billing Application!")
            print("=" * 75 + "\n")
            break

        else:
            print("\nInvalid choice. Please enter 1, 2, 3, or 4.")

# Execute the program
if __name__ == "__main__":
    main()
```

OUTPUT:


Electricity bill calculation application
2m · +472 · Auto
The final electricity bill must present all values clearly.
Develop the final Python application to:
Professional formatting
Real-world applicability:
Implements utility billing structure
Multiple customer categories
Progressive slab-based pricing
Government-mandated duty
Production-ready error handling
4. User interface
Interactive mode for custom calculations
Demonstration mode with examples
Program analysis view
Input validation and error handling
The application is ready to use. Run it with:
python task5-3.3.py
It presents all bill components clearly and includes
1 File
Undo
Keep
Review
New update available
Later
Install Now
task5-3.3.py
AI_LAB
ASS1
add.py
data.csv
sample_file.txt
sort_comparison.py
system_errors.log
task1-2.3.py
TASK1-3.3.PY
task2-2.3.py
task2-3.3.py
task3-2.3.py
task3-3.3.py
task4-2.3.py
task4-3.3.py
task5-3.3.py
This is the complete, production-ready electricity billin
and displays comprehensive bills with all charge componen
Components:
- EC (Energy Charges): Based on units consumed and custom
- FC (Fixed Charges): Monthly fixed charges based on cust
- CC (Customer Charges3): Fixed monthly charge per custom
- ED (Electricity Duty): Percentage-based tax on energy c
Total Bill Formula: Total Bill = EC + FC + CC + ED
Analysis:
Problems
Output
Terminal
Python
OPTIONS:
1. Calculate bill (Interactive mode)
2. View demonstration examples
3. View program analysis
4. Exit
Enter your choice (1-4):
21°C
Sunny
Search
ENG
IN
10:53
22-01-2026