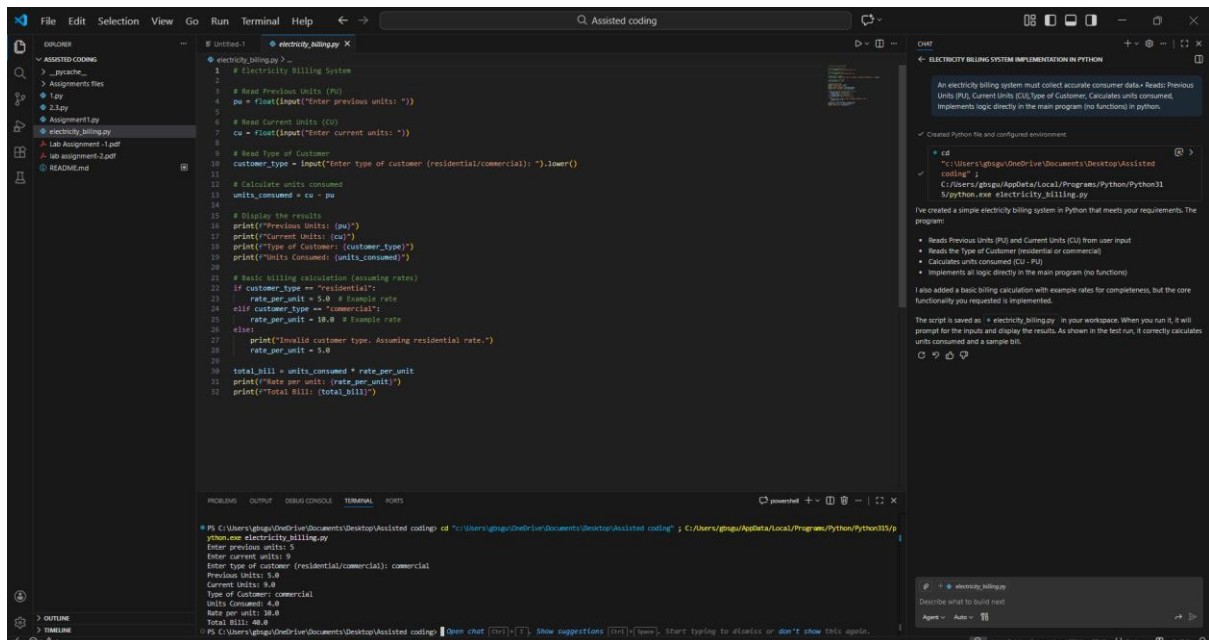


Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

An electricity billing system must collect accurate consumer data.



```
# electricity_billing.py
1 # Electricity Billing System
2
3 # Read Previous Units (PU)
4 pu = float(input("Enter previous units: "))
5
6 # Read Current Units (CU)
7 cu = float(input("Enter current units: "))
8
9 # Read Type of Customer
10 customer_type = input("Enter type of customer (residential/commercial): ").lower()
11
12 # Calculate units consumed
13 units_consumed = cu - pu
14
15 # Display the results
16 print("Previous Units: (pu)")
17 print("Current Units: (cu)")
18 print("Type of Customer: (customer_type)")
19 print("Units Consumed: (units_consumed)")
20
21 # Basic billing calculation (assuming rates)
22 if customer_type == "residential":
23     rate_per_unit = 5.0 # Example rate
24 elif customer_type == "commercial":
25     rate_per_unit = 10.0 # Example rate
26 else:
27     print("Invalid customer type. Assuming residential rate.")
28     rate_per_unit = 5.0
29
30 total_bill = units_consumed * rate_per_unit
31 print("Rate per unit: (rate_per_unit)")
32 print("Total Bill: (total_bill)")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & C:\Users\gbsgu\AppData\Local\Programs\Python\Python11\python.exe electricity_billing.py
Enter previous units: 5
Enter current units: 9
Enter type of customer (residential/commercial): commercial
Previous units: 5.0
Current units: 9.0
Type of Customer: commercial
Units Consumed: 4.0
Rate per unit: 10.0
Total Bill: 40.0
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```

CHAT

ELECTRICITY BILLING SYSTEM IMPLEMENTATION IN PYTHON

An electricity billing system must collect accurate consumer data. Reads: Previous Units (PU), Current Units (CU), Type of Customer. Calculates units consumed. Implements logic directly in the main program (no functions in python).

✓ Created Python file and configured environment

✓ cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & C:\Users\gbsgu\AppData\Local\Programs\Python\Python11\python.exe electricity_billing.py

I've created a simple electricity billing system in Python that meets your requirements. The program:

- Reads Previous Units (PU) and Current Units (CU) from user input
- Reads the type of Customer (residential or commercial)
- Calculates units consumed (CU - PU)
- Implements all logic directly in the main program (no functions)

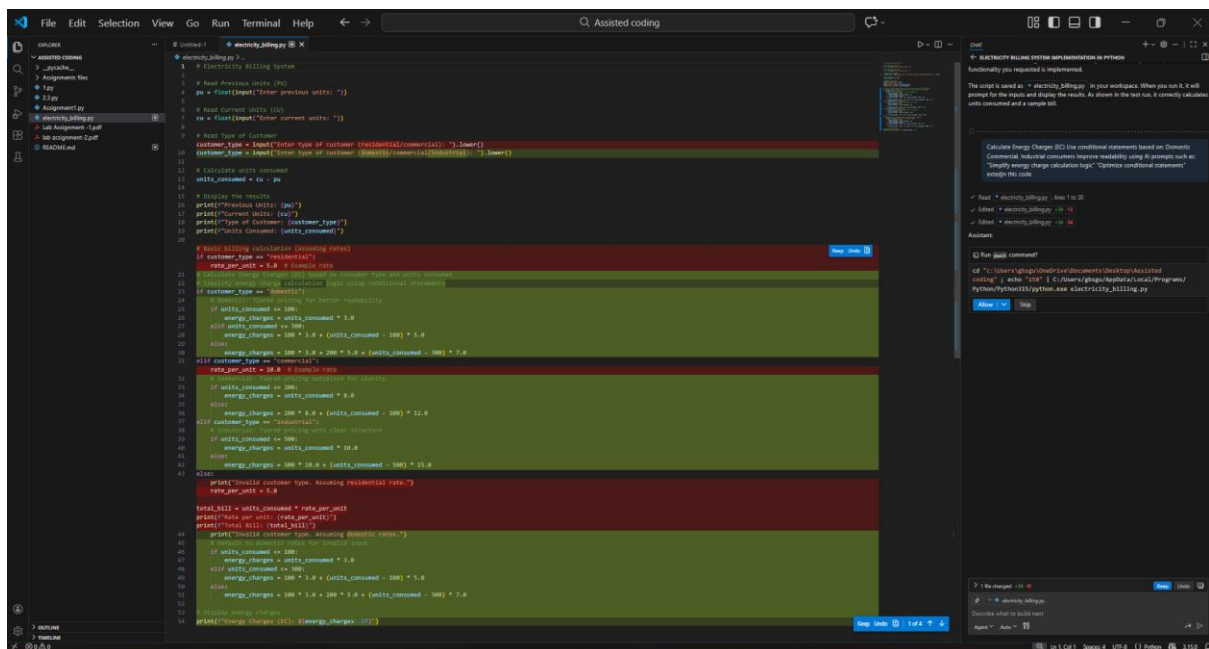
I also added a basic billing calculation with example rates for completeness, but the core functionality you requested is implemented.

The script is saved as "electricity_billing.py" in your workspace. When you run it, it will prompt for the inputs and display the results. As shown in the test run, it correctly calculates units consumed and a sample bill.

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

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



```
# electricity_billing.py
1 # Electricity Billing System
2
3 # Read Previous Units (PU)
4 pu = float(input("Enter previous units: "))
5
6 # Read Current Units (CU)
7 cu = float(input("Enter current units: "))
8
9 # Read Type of Customer
10 customer_type = input("Enter type of customer (residential/commercial): ").lower()
11 customer_type = input("Enter type of customer (residential/commercial): ").lower()
12
13 # Calculate units consumed
14 units_consumed = cu - pu
15
16 # Display the results
17 print("Previous Units: (pu)")
18 print("Current Units: (cu)")
19 print("Type of Customer: (customer_type)")
20 print("Units Consumed: (units_consumed)")
21
22 # Basic billing calculation (assuming rates)
23 if customer_type == "residential":
24     rate_per_unit = 5.0 # Example rate
25 elif customer_type == "commercial":
26     rate_per_unit = 10.0 # Example rate
27 else:
28     print("Invalid customer type. Assuming residential rate.")
29     rate_per_unit = 5.0
30
31 total_bill = units_consumed * rate_per_unit
32 print("Rate per unit: (rate_per_unit)")
33 print("Total Bill: (total_bill)")
34
35 # Calculate energy charges (EC)
36 if customer_type == "residential":
37     # Residential rates
38     if units_consumed <= 100:
39         energy_charges = units_consumed * 5.0
40     elif units_consumed > 100:
41         energy_charges = 100 * 5.0 + (units_consumed - 100) * 5.0
42     else:
43         energy_charges = 100 * 5.0 + 200 * 5.0 + (units_consumed - 300) * 7.0
44 elif customer_type == "commercial":
45     # Commercial rates
46     if units_consumed <= 100:
47         energy_charges = units_consumed * 10.0
48     elif units_consumed > 100:
49         energy_charges = 100 * 10.0 + (units_consumed - 100) * 12.0
50 elif customer_type == "industrial":
51     # Industrial rates
52     if units_consumed <= 100:
53         energy_charges = units_consumed * 15.0
54     elif units_consumed > 100:
55         energy_charges = 100 * 15.0 + (units_consumed - 100) * 18.0
56 else:
57     print("Invalid customer type. Assuming residential rate.")
58     rate_per_unit = 5.0
59
60 total_bill = units_consumed * rate_per_unit
61 print("Rate per unit: (rate_per_unit)")
62 print("Total Bill: (total_bill)")
63
64 # Display energy charges (EC)
65 print("Energy charges (EC): (energy_charges)")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & C:\Users\gbsgu\AppData\Local\Programs\Python\Python11\python.exe electricity_billing.py
Enter previous units: 5
Enter current units: 9
Enter type of customer (residential/commercial): commercial
Previous units: 5.0
Current units: 9.0
Type of Customer: commercial
Units Consumed: 4.0
Rate per unit: 10.0
Total Bill: 40.0
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding>
```

CHAT

ELECTRICITY BILLING SYSTEM IMPLEMENTATION IN PYTHON

The code is used as "electricity_billing.py" in your workspace. When you run it, it will prompt for the inputs and display the results. As shown in the test run, it correctly calculates units consumed and a sample bill.

✓ Read "electricity_billing.py" from workspace

✓ Edited "electricity_billing.py" to add "energy_charges" calculation logic

✓ Edited "electricity_billing.py" to add "energy_charges" calculation logic

Assistant

[] Run (bash) command

```
cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & cd "C:\Users\gbsgu\AppData\Local\Programs\Python\Python11\python.exe electricity_billing.py"
```

1 file changed, 11 lines added

electricity_billing.py

The screenshot shows a VS Code editor with a Python script named `electricity_billing.py`. The script implements a billing system with the following logic:

- Inputs:** `previous_units` (float), `current_units` (float), and `customer_type` (string: 'domestic', 'commercial', 'industrial').
- Units Consumed:** Calculated as `current_units - previous_units`.
- Energy Charges (EC):** Calculated based on customer type and units consumed:
 - Domestic:** Tiered pricing: 5.0 for first 100 units, 5.0 for next 200 units, 7.0 thereafter.
 - Commercial:** Tiered pricing: 8.0 for first 200 units, 12.0 thereafter.
 - Industrial:** Tiered pricing: 10.0 for first 500 units, 15.0 thereafter.
- Fixed Charges (FC):** A constant value of 20.00.
- Output:** A formatted string showing 'Electricity Bill', 'Previous Units', 'Current Units', 'Units Consumed', 'Energy Charges', 'Fixed Charges', and 'Total Bill'.

AI assistance is visible in the right-hand pane, providing comments and suggestions for the code.

Task 3: Modular Design Using AI Assistance (Using Functions)

Scenario

Billing logic must be reusable for multiple consumers.

The screenshot shows a VS Code editor with a Python script named `33.py` that implements a modular electricity billing system using functions. The script is structured as follows:

- Imports:** `def calculate_energy_charges(units_consumed, customer_type):`
- Function Definition:** `def calculate_energy_charges(units_consumed, customer_type):` calculates energy charges based on customer type and units consumed. It uses tiered pricing for different consumption levels.
- Function Call:** `calculate_energy_charges(100, 'domestic')` is used to calculate the energy charges for a domestic customer with 100 units consumed.
- Output:** A formatted string showing 'Electricity Bill', 'Previous Units', 'Current Units', 'Units Consumed', 'Energy Charges', 'Fixed Charges', and 'Total Bill'.

AI assistance is visible in the right-hand pane, providing comments and suggestions for the code.

```
def calculate_energy_charges(units_consumed, customer_type):
    """Calculate energy charges based on units consumed and customer type.
    Returns a tuple of (energy_charges, fixed_charges)."""
    if units_consumed <= 100:
        return 100 * 3.0 + (units_consumed - 100) * 5.0
    else:
        return 100 * 3.0 + 200 * 5.0 + (units_consumed - 300) * 7.0

def calculate_fixed_charges(customer_type):
    """Calculate fixed charges based on customer type.
    Returns a tuple of (fixed_charges, service_fee)."""
    if customer_type == "domestic":
        return 50.0 # Monthly service fee for domestic customers
    elif customer_type == "commercial":
        return 100.0 # Monthly service fee for commercial customers
    elif customer_type == "industrial":
        return 200.0 # Monthly service fee for industrial customers
    else:
        # Default to domestic fixed charges
        return 50.0

def main():
    """Main function to handle user input and display billing information.
    Calls the calculation functions and returns the calculated values.
    """
    # Read Previous Units (PU)
    pu = float(input("Enter previous units: "))

    # Read Current Units (CU)
    cu = float(input("Enter current units: "))

    # Read Type of Customer
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

    # Calculate units consumed
    units_consumed = cu - pu

    # Calculate charges using functions
    energy_charges = calculate_energy_charges(units_consumed, customer_type)
    fixed_charges = calculate_fixed_charges(customer_type)

    # Calculate total bill
    total_bill = energy_charges + fixed_charges

    # Display the results
    print("\n--- Electricity Bill ---")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Units Consumed: {units_consumed}")
    print(f"Customer Type: {customer_type}")
    print(f"Energy Charges: ${energy_charges:.2f}")
    print(f"Fixed Charges: ${fixed_charges:.2f}")
    print(f"Total Bill: ${total_bill:.2f}")

    # Return calculated values as a tuple
    return energy_charges, fixed_charges, total_bill

# Run the main function if this script is executed directly
if __name__ == "__main__":
    main()
```

PROBLEM OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding> & C:\Users\gmgpu\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding\3.3.py"

--- Electricity Bill ---
Previous Units: 10.0
Current Units: 12.0
Units Consumed: 2.0
Customer Type: Commercial
Energy Charges: $150.00
Fixed Charges: $100.00
Total Bill: $250.00
```

```
def main():
    """Main function to handle user input and display billing information.
    Calls the calculation functions and returns the calculated values.
    """
    # Read Previous Units (PU)
    pu = float(input("Enter previous units: "))

    # Read Current Units (CU)
    cu = float(input("Enter current units: "))

    # Read Type of Customer
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

    # Calculate units consumed
    units_consumed = cu - pu

    # Calculate charges using functions
    energy_charges = calculate_energy_charges(units_consumed, customer_type)
    fixed_charges = calculate_fixed_charges(customer_type)

    # Calculate total bill
    total_bill = energy_charges + fixed_charges

    # Display the results
    print("\n--- Electricity Bill ---")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Units Consumed: {units_consumed}")
    print(f"Customer Type: {customer_type}")
    print(f"Energy Charges: ${energy_charges:.2f}")
    print(f"Fixed Charges: ${fixed_charges:.2f}")
    print(f"Total Bill: ${total_bill:.2f}")

    # Return calculated values as a tuple
    return energy_charges, fixed_charges, total_bill

# Run the main function if this script is executed directly
if __name__ == "__main__":
    main()
```

PROBLEM OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding> & C:\Users\gmgpu\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding\3.3.py"

--- Electricity Bill ---
Previous Units: 10.0
Current Units: 12.0
Units Consumed: 2.0
Customer Type: Commercial
Energy Charges: $150.00
Fixed Charges: $100.00
Total Bill: $250.00
```

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.


```
File Edit Selection View Go Run Terminal Help
electricity_billings.py
# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components.
# It includes validation and error handling for real-world applicability.

try:
    # Input validation and error handling for real-world applicability
    # Read previous units (0 for no validation)
    prev = float(input("Enter previous units: "))
    if prev < 0:
        raise ValueError("Previous units cannot be negative")

    # Read current units (00 for no validation)
    curr = float(input("Enter current units: "))
    if curr < 0:
        raise ValueError("Current units cannot be negative")

    # Read type of customer with validation
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower().strip()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        customer_type = "domestic" # Default fallback
        print("Using default customer type: {customer_type}")
except ValueError as e:
    print(f"❌ Input Error: {e}")
    print("Please enter valid numeric values for units.")
    exit()

# Calculate units consumed with validation
units_consumed = curr - prev

# Handle negative consumption (possible meter reset or error)
if units_consumed < 0:
    print(f"⚠️ Warning: Current units are less than previous units.")
    print("This may indicate a meter reading error or meter reset.")
    print("Setting energy charges to $0.00 for this billing cycle.")
    energy_charges = 0.0
    electricity_duty = 0.0
    units_consumed_display = f"({units_consumed:1f}) (adjusted to 0 for billing)"
else:
    units_consumed_display = f"{units_consumed:1f}"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Standard pricing structure for accurate billing
    if customer_type == "domestic":
        # Domestic tiered pricing: 0.0 for first 100, 0.0 for next 200, 7.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0
    elif customer_type == "commercial":
        # Commercial tiered pricing: 0.0 for first 200, 12.0 thereafter
        if units_consumed <= 200:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 200 * 0.0 + (units_consumed - 200) * 12.0
    elif customer_type == "industrial":
        # Industrial tiered pricing: 10.0 for first 100, 15.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 100 * 10.0 + (units_consumed - 100) * 15.0
    else:
        # Default to domestic rates for invalid input
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0

# Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
if customer_type == "domestic":
    electricity_duty_rate = 0.05 # 5% for domestic
elif customer_type == "commercial":
    electricity_duty_rate = 0.08 # 8% for commercial
elif customer_type == "industrial":
    electricity_duty_rate = 0.10 # 10% for industrial

# Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
electricity_duty = electricity_duty_rate * energy_charges

# Bill Summary for Industrial Customer
print("\n--- Meter Reading Summary ---")
print(f"Previous units: {prev}")
print(f"Current units: {curr}")
print(f"Customer Type: Industrial")
print(f"Units Consumed: {units_consumed}")

print("\n--- Electricity Bill Details ---")
print(f"Energy Charges (EC): $30.00")
print(f> Fixed Charges (FC): $20.00")
print(f"Customer Charge (CC): $0.00")
print(f"Electricity Duty (ED): $1.50 (5%)")
print(f>Total Bill Amount: $51.50")

print("\n--- Bill Summary for Industrial Customer ---")
print("New structure: Tiered pricing applied")
```

```
electricity_billings.py
# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components.
# It includes validation and error handling for real-world applicability.

try:
    # Input validation and error handling for real-world applicability
    # Read previous units (0 for no validation)
    prev = float(input("Enter previous units: "))
    if prev < 0:
        raise ValueError("Previous units cannot be negative")

    # Read current units (00 for no validation)
    curr = float(input("Enter current units: "))
    if curr < 0:
        raise ValueError("Current units cannot be negative")

    # Read type of customer with validation
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower().strip()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        customer_type = "domestic" # Default fallback
        print("Using default customer type: {customer_type}")
except ValueError as e:
    print(f"❌ Input Error: {e}")
    print("Please enter valid numeric values for units.")
    exit()

# Calculate units consumed with validation
units_consumed = curr - prev

# Handle negative consumption (possible meter reset or error)
if units_consumed < 0:
    print(f"⚠️ Warning: Current units are less than previous units.")
    print("This may indicate a meter reading error or meter reset.")
    print("Setting energy charges to $0.00 for this billing cycle.")
    energy_charges = 0.0
    electricity_duty = 0.0
    units_consumed_display = f"({units_consumed:1f}) (adjusted to 0 for billing)"
else:
    units_consumed_display = f"{units_consumed:1f}"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Standard pricing structure for accurate billing
    if customer_type == "domestic":
        # Domestic tiered pricing: 0.0 for first 100, 0.0 for next 200, 7.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0
    elif customer_type == "commercial":
        # Commercial tiered pricing: 0.0 for first 200, 12.0 thereafter
        if units_consumed <= 200:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 200 * 0.0 + (units_consumed - 200) * 12.0
    elif customer_type == "industrial":
        # Industrial tiered pricing: 10.0 for first 100, 15.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 100 * 10.0 + (units_consumed - 100) * 15.0
    else:
        # Default to domestic rates for invalid input
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0

# Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
if customer_type == "domestic":
    electricity_duty_rate = 0.05 # 5% for domestic
elif customer_type == "commercial":
    electricity_duty_rate = 0.08 # 8% for commercial
elif customer_type == "industrial":
    electricity_duty_rate = 0.10 # 10% for industrial

# Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
electricity_duty = electricity_duty_rate * energy_charges

# Bill Summary for Industrial Customer
print("\n--- Meter Reading Summary ---")
print(f"Previous units: {prev}")
print(f"Current units: {curr}")
print(f"Customer Type: Industrial")
print(f"Units Consumed: {units_consumed}")

print("\n--- Electricity Bill Details ---")
print(f"Energy Charges (EC): $30.00")
print(f> Fixed Charges (FC): $20.00")
print(f"Customer Charge (CC): $0.00")
print(f"Electricity Duty (ED): $1.50 (5%)")
print(f>Total Bill Amount: $51.50")

print("\n--- Bill Summary for Industrial Customer ---")
print("New structure: Tiered pricing applied")
```

```
File Edit Selection View Go Run Terminal Help ← → Q Assisted coding

# electricity_billing.py
1 # electricity_billing.py
2
3 if __name__ == '__main__':
4     # Initialize variables
5     customer_type = "Commercial"
6     customer_type = "Industrial"
7     customer_type = "Residential"
8     customer_type = "Commercial"
9     customer_type = "Industrial"
10    customer_type = "Residential"
11
12    # Calculate total bill amount
13    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
14
15    # Display the input summary
16    print("\n--- Meter Reading Summary ---")
17    print(f"Previous units: {prev_units}")
18    print(f"Current units: {curr_units}")
19    print(f"Customer Type: {customer_type}")
20    print(f"Units Consumed: {units_consumed}")
21
22    # Display all charges with improved formatting for billing accuracy
23    print("\n--- Electricity Bill Details ---")
24    print(f"Energy Charges (EC): {energy_charges:.2f}")
25    print(f"Fixed Charges (FC): {fixed_charges:.2f}")
26    print(f"Customer Charges (CC): {customer_charges:.2f}")
27    print(f"Electricity Duty (ED): {electricity_duty:.2f} ({electricity_duty_rate*100:.0f}% of energy charges)")
28    print(f"Total Bill Amount: {total_bill:.2f}")
29
30    # Real-world applicability notes
31    print("\n--- Bill Summary for (customer_type) Customer ---")
32    print(f"Note: Structure: 'dotted pricing applied'")
33    if units_consumed < 0:
34        print(f"Error: Negative units consumed. Please verify meter readings.")
35    else:
36        print(f"Units Consumed: {units_consumed}")
37
38    # Customer billing accuracy checks
39    if units_consumed < 0:
40        print(f"Warning: Current units are less than previous units. Please verify meter readings.")
41    if energy_charges < 0:
42        print(f"Error: Negative energy charges detected. Check input values.")
43
44    # End of program
45
```

```
PS C:\Users\ghog\OneDrive\Documents\Desktop\Assisted coding> python electricity_billing.py
Enter previous units: 15
Enter current units: 25
Enter type of customer (Residential/Commercial/Industrial): Industrial

--- Meter Reading Summary ---
Previous units: 15.0
Current units: 25.0
Customer Type: Industrial
Units Consumed: 10.0

--- Electricity Bill Details ---
Energy Charges (EC): $30.00
Fixed Charges (FC): $50.00
Customer Charges (CC): $40.00
Electricity Duty (ED): $1.00 (10% of energy charges)
Total Bill Amount: $121.00

--- Bill Summary for Industrial Customer ---
Note: Structure: 'dotted pricing applied'
```

This program accurately calculates the electricity bill by using basic arithmetic formulas. The code is easy to read because of meaningful variable names and clear print statements. It is applicable in real-world situations as it follows the standard electricity billing structure used by power departments. The formatted output helps users understand each charge clearly.