

ASSINMENT 3.3

NAME:K.Naga Chaitanya

HALLTICKET:2303A51292

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

An electricity billing system must collect accurate consumer data.

Task Description

Use an AI tool (GitHub Copilot / Gemini) to 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)

PROMPT: Write a program to read electricity meter readings, calculate units consumed, display details, and validate the input.

The screenshot displays an AI code generation interface with a dark theme. On the left, a sidebar shows a list of agents, including 'AI-generated electricity...' and 'Gemini Python word fre...'. The main area is divided into two panels. The left panel, titled 'AI-generated electricity billing lo...', contains a prompt and a list of requirements: 'All logic is implemented directly in the main program (no functions)', 'Includes basic validation to check for negative or zero consumption', and 'How it works: 1. Prompts for Previous Units (PU), 2. Prompts for Current Units (CU), 3. Prompts for Type of Customer, 4. Calculates units consumed = CU - PU, 5. Displays all consumer details including the calculated units consumed'. Below this, it says 'You can run it with: python electricity_billing.py' and 'The program will prompt for inputs and display the results, including the calculated units consumed.' The right panel shows the generated Python code for 'electricity_billing.py'. The code includes comments and prompts for 'Previous Units (PU)', 'Current Units (CU)', and 'Type of Customer', followed by calculations for 'units_consumed' and a validation check for negative values.

```
1 # Electricity Billing System - Consumer Details Reader
2 # This program reads consumer data and calculates units consumed
3
4 # Read Previous Units (PU)
5 print("=== Electricity Billing System ===")
6 print()
7 previous_units = float(input("Enter Previous Units (PU): "))
8
9 # Read Current Units (CU)
10 current_units = float(input("Enter Current Units (CU): "))
11
12 # Read Type of Customer
13 customer_type = input("Enter Type of Customer (e.g., Residential, Commercial, Industrial): ")
14
15 # Calculate units consumed
16 units_consumed = current_units - previous_units
17
18 # Display results
19 print()
20 print("=== Consumer Details === (variable) previous_units: float")
21 print(f"Previous Units (PU): {previous_units}")
22 print(f"Current Units (CU): {current_units}")
23 print(f"Type of Customer: {customer_type}")
24 print(f"Units Consumed: {units_consumed}")
25
26 # Validation check
27 if units_consumed < 0:
28     print()
29     print("Warning: Current Units is less than Previous Units. Please verify the readings.")
30 elif units_consumed == 0:
31     print()
32     print("Note: No units consumed in this billing period.")
33
```

OUTPUT:

The screenshot shows a code editor with a dark theme. On the left, there's a sidebar with 'Agents' and 'AI-generated electricity billing logic'. The main editor area displays the output of a Python program. The program prompts the user for previous units, current units, and customer type. It then calculates the units consumed and displays the results. The output shows that the current units (23) are less than the previous units (45), resulting in a negative value for units consumed (-22.0). A warning message is displayed: 'Warning: Current Units is less than Previous Units. Please verify the readings.'

```
PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/bin/python.exe c:/Users/Administrator/Desktop/aiasscode/electricity_billing.py
=== Electricity Billing System ===

Enter Previous Units (PU): 45
Enter Current Units (CU): 23
Enter Type of Customer (e.g., Residential, Commercial, Industrial): 2

=== Consumer Details ===
Previous Units (PU): 45.0
Current Units (CU): 23.0

=== Consumer Details ===
Previous Units (PU): 45.0
Current Units (CU): 23.0
Current Units (CU): 23.0
Type of Customer: 2
Units Consumed: -22.0

Warning: Current Units is less than Previous Units. Please verify the readings.
Type of Customer: 2
Units Consumed: -22.0

Warning: Current Units is less than Previous Units. Please verify the readings.
PS C:\Users\Administrator\Desktop\aiasscode>
```

EXPLANATION: This program reads the previous and current electricity meter readings from the user, along with the type of customer. It calculates the number of units consumed by subtracting the previous reading from the current reading. The program then displays all the consumer details and checks for errors by warning the user if the current reading is less than the previous reading or noting if no units were consumed

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

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

Task Description

Review the AI-generated code from Task 1 and extend it to:

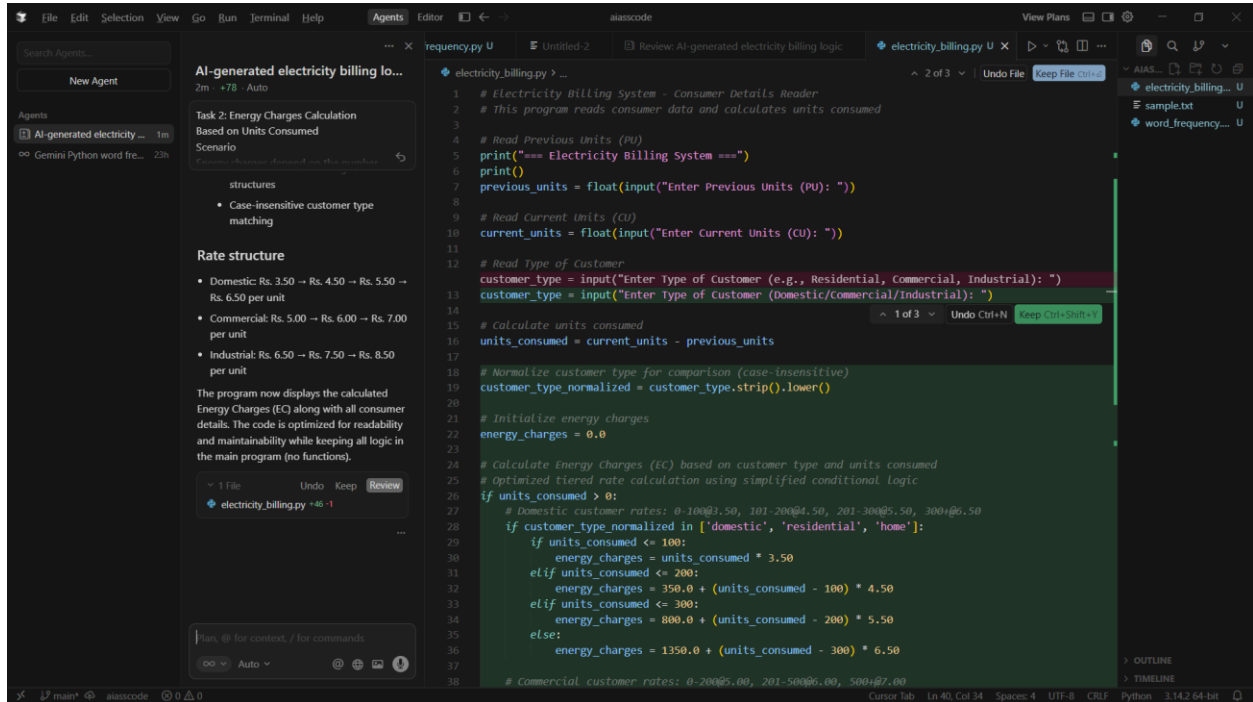
- Calculate Energy Charges (EC)
- Use conditional statements based on:
 - o Domestic
 - o Commercial
 - o Industrial consumers
- Improve readability using AI prompts such as:

o “Simplify energy charge calculation logic”

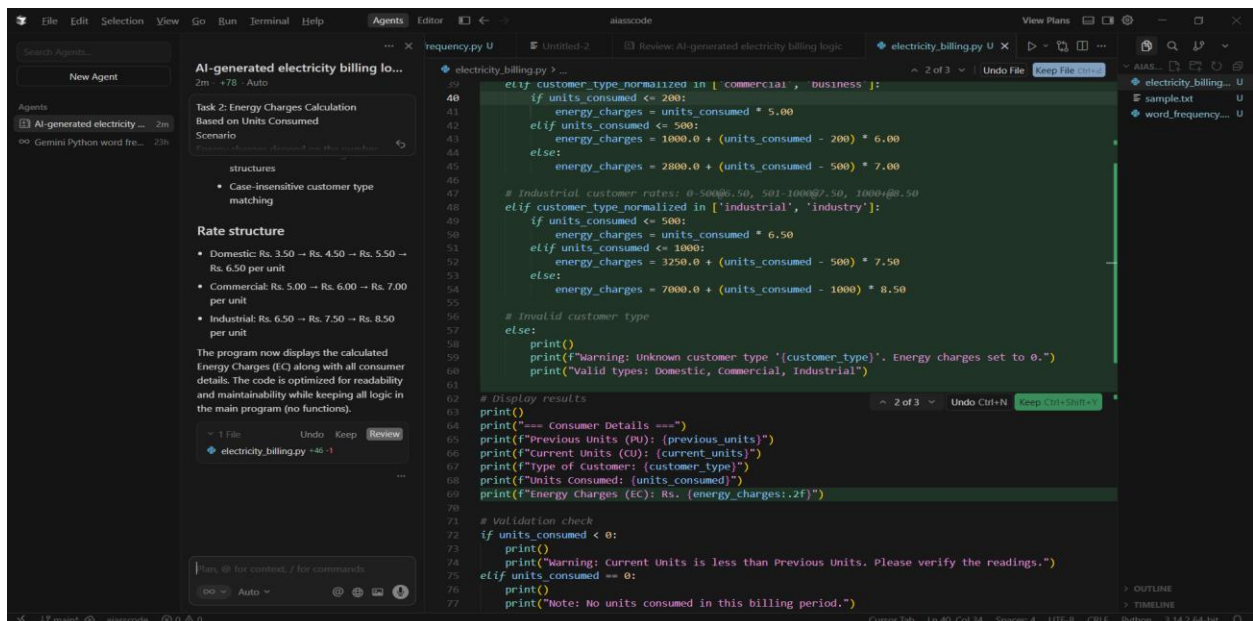
o “Optimize conditional statements”

PROMPT:

Write a program to calculate electricity units and energy charges based on customer type and slab rates, and display the bill details.



```
1  # Electricity Billing System - consumer Details Reader
2  # This program reads consumer data and calculates units consumed
3
4  # Read Previous Units (PU)
5  print("=== Electricity Billing System ===")
6  print()
7  previous_units = float(input("Enter Previous Units (PU): "))
8
9  # Read Current Units (CU)
10 current_units = float(input("Enter Current Units (CU): "))
11
12 # Read Type of Customer
13 customer_type = input("Enter Type of Customer (e.g., Residential, Commercial, Industrial): ")
14 customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")
15
16 # Calculate units consumed
17 units_consumed = current_units - previous_units
18
19 # Normalize customer type for comparison (case-insensitive)
20 customer_type_normalized = customer_type.strip().lower()
21
22 # Initialize energy charges
23 energy_charges = 0.0
24
25 # Calculate Energy Charges (EC) based on customer type and units consumed
26 # Optimized tiered rate calculation using simplified conditional logic
27 if units_consumed > 0:
28     # Domestic customer rates: 0-100@3.50, 101-200@4.50, 201-300@5.50, 300+@6.50
29     if customer_type_normalized in ['domestic', 'residential', 'home']:
30         if units_consumed <= 100:
31             energy_charges = units_consumed * 3.50
32         elif units_consumed <= 200:
33             energy_charges = 350.0 + (units_consumed - 100) * 4.50
34         elif units_consumed <= 300:
35             energy_charges = 800.0 + (units_consumed - 200) * 5.50
36         else:
37             energy_charges = 1350.0 + (units_consumed - 300) * 6.50
38     # Commercial customer rates: 0-200@5.00, 201-500@6.00, 500+@7.00
```



```
39 elif customer_type_normalized in ['commercial', 'business']:
40     if units_consumed <= 200:
41         energy_charges = units_consumed * 5.00
42     elif units_consumed <= 500:
43         energy_charges = 1000.0 + (units_consumed - 200) * 6.00
44     else:
45         energy_charges = 2800.0 + (units_consumed - 500) * 7.00
46
47 # Industrial customer rates: 0-500@6.50, 501-1000@7.50, 1000+@8.50
48 elif customer_type_normalized in ['industrial', 'industry']:
49     if units_consumed <= 500:
50         energy_charges = units_consumed * 6.50
51     elif units_consumed <= 1000:
52         energy_charges = 3250.0 + (units_consumed - 500) * 7.50
53     else:
54         energy_charges = 7000.0 + (units_consumed - 1000) * 8.50
55
56 # Invalid customer type
57 else:
58     print()
59     print(f"Warning: Unknown customer type '{customer_type}'. Energy charges set to 0.")
60     print("Valid types: Domestic, commercial, Industrial")
61
62 # Display results
63 print()
64 print("== Consumer Details ==")
65 print(f"Previous Units (PU): {previous_units}")
66 print(f"Current Units (CU): {current_units}")
67 print(f"Type of Customer: {customer_type}")
68 print(f"Units Consumed: {units_consumed}")
69 print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")
70
71 # Validation check
72 if units_consumed < 0:
73     print()
74     print("Warning: Current Units is less than Previous Units. Please verify the readings.")
75 elif units_consumed == 0:
76     print()
77     print("Note: No units consumed in this billing period.")
```

The screenshot shows the VS Code interface with the following components:

- Left Sidebar:**
 - Agents Panel:** Shows 'AI-generated electricity' as the active agent.
 - Task 2: Energy Charges Calculation:** Based on Units Consumed Scenario.
 - Rate structure:**
 - Domestic: Rs. 3.50 → Rs. 4.50 → Rs. 5.50 → Rs. 6.50 per unit
 - Commercial: Rs. 5.00 → Rs. 6.00 → Rs. 7.00 per unit
 - Industrial: Rs. 6.50 → Rs. 7.50 → Rs. 8.50 per unit
- Main Editor:**
 - electricity_billing.py:**

```

def customer_type_normalized(in ['commercial', 'business']):
    if units_consumed <= 200:
        energy_charges = units_consumed * 5.00
    elif units_consumed <= 500:
        energy_charges = 1000.0 + (units_consumed - 200) * 6.00
    else:
        energy_charges = 2800.0 + (units_consumed - 500) * 7.00

```
- Bottom Terminal:**

```

PS C:\Users\Administrator\Desktop\aiasscode> & C:/Users/Administrator/AppData/Local/Python/bin/python.exe c:/Users/Administrator/Desktop/aiasscode/electricity_billing.py
=== Electricity Billing System ===

Enter Previous Units (PU): 34
Enter Current Units (CU): 21
Enter Type of Customer (Domestic/Commercial/Industrial): 7

=== Consumer Details ===
Previous Units (PU): 34.0
Current Units (CU): 21.0
Type of Customer: 7
Units Consumed: -13.0
Energy Charges (EC): Rs. 0.00

Warning: Current Units is less than Previous Units. Please verify the readings.
PS C:\Users\Administrator\Desktop\aiasscode>

```

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

Billing logic must be reusable for multiple consumers.

Use AI assistance to generate a Python program that:

- Uses user-defined functions to:
 - o Calculate Energy Charges
 - o Calculate Fixed Charges
- Returns calculated values
- Includes meaningful comments

File Edit Selection View Go Run Terminal Help

Agents Editor

aiasscode

View Plans

Search Agents...

New Agent

Agents

- AI-generated electricity ... 2m
- Gemini Python word fre... 23h

AI-generated electricity billing lo...
3m +137 Auto

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

- Fixed Charges (FC)
- Total Charges (EC + FC)
- Maintains validation checks

Benefits of modular design

- Reusability: Functions can be called for multiple consumers
- Maintainability: Rate changes only require updating the functions
- Readability: Clear separation of concerns with meaningful comments
- Testability: Functions can be tested independently
- Documentation: Docstrings explain purpose, parameters, and return values

The program is now modular and ready to handle billing for multiple consumers efficiently.

> 1 File

Review

Plan, @ for context, / for commands

Auto

electricity_billing.py U

Unfiled-2

Review: AI-generated electricity billing logic

electricity_billing.py U X

AIASSCODE

- electricity_billing... U
- sample.txt U
- word_frequency... U

OUTLINE

TIMELINE

```
1 # Electricity Billing System - Modular Design with Functions
2 # This program reads consumer data and calculates billing charges using reusable functions
3
4 def calculate_energy_charges(units_consumed, customer_type):
5     """
6     Calculate Energy Charges (EC) based on units consumed and customer type.
7
8     Uses tiered rate structure:
9     - Domestic: 0-100@3.50, 101-200@4.50, 201-300@5.50, 300+@6.50
10    - Commercial: 0-200@5.00, 201-500@6.00, 500+@7.00
11    - Industrial: 0-500@6.50, 501-1000@7.50, 1000+@8.50
12
13    Args:
14        units_consumed (float): Number of units consumed
15        customer_type (str): Type of customer (Domestic/Commercial/Industrial)
16
17    Returns:
18        float: Calculated energy charges in rupees
19    """
20    # Normalize customer type for case-insensitive comparison
21    customer_type_normalized = customer_type.strip().lower()
22
23    # Return 0 if no units consumed or invalid input
24    if units_consumed <= 0:
25        return 0.0
26
27    # Calculate energy charges for Domestic customers
28    if customer_type_normalized in ['domestic', 'residential', 'home']:
29        if units_consumed <= 100:
30            return units_consumed * 3.50
31        elif units_consumed <= 200:
32            return 350.0 + (units_consumed - 100) * 4.50
33        elif units_consumed <= 300:
34            return 800.0 + (units_consumed - 200) * 5.50
35        else:
36            return 1350.0 + (units_consumed - 300) * 6.50
37
38    # Calculate energy charges for Commercial customers
39    elif customer_type_normalized in ['commercial', 'business']:
```

Cursor Tab | Ln 6, Col 63 | Spaces: 4 | UTF-8 | CRLF | Python | 3.14.2-64-bit

File Edit Selection View Go Run Terminal Help

Agents Editor

aiasscode

View Plans

Search Agents...

New Agent

Agents

- AI-generated electricity ... 3m
- Gemini Python word fre... 23h

AI-generated electricity billing lo...
3m +137 Auto

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

- Fixed Charges (FC)
- Total Charges (EC + FC)
- Maintains validation checks

Benefits of modular design

- Reusability: Functions can be called for multiple consumers
- Maintainability: Rate changes only require updating the functions
- Readability: Clear separation of concerns with meaningful comments
- Testability: Functions can be tested independently
- Documentation: Docstrings explain purpose, parameters, and return values

The program is now modular and ready to handle billing for multiple consumers efficiently.

> 1 File

Review

Plan, @ for context, / for commands

Auto

electricity_billing.py U

Unfiled-2

Review: AI-generated electricity billing logic

electricity_billing.py U X

AIASSCODE

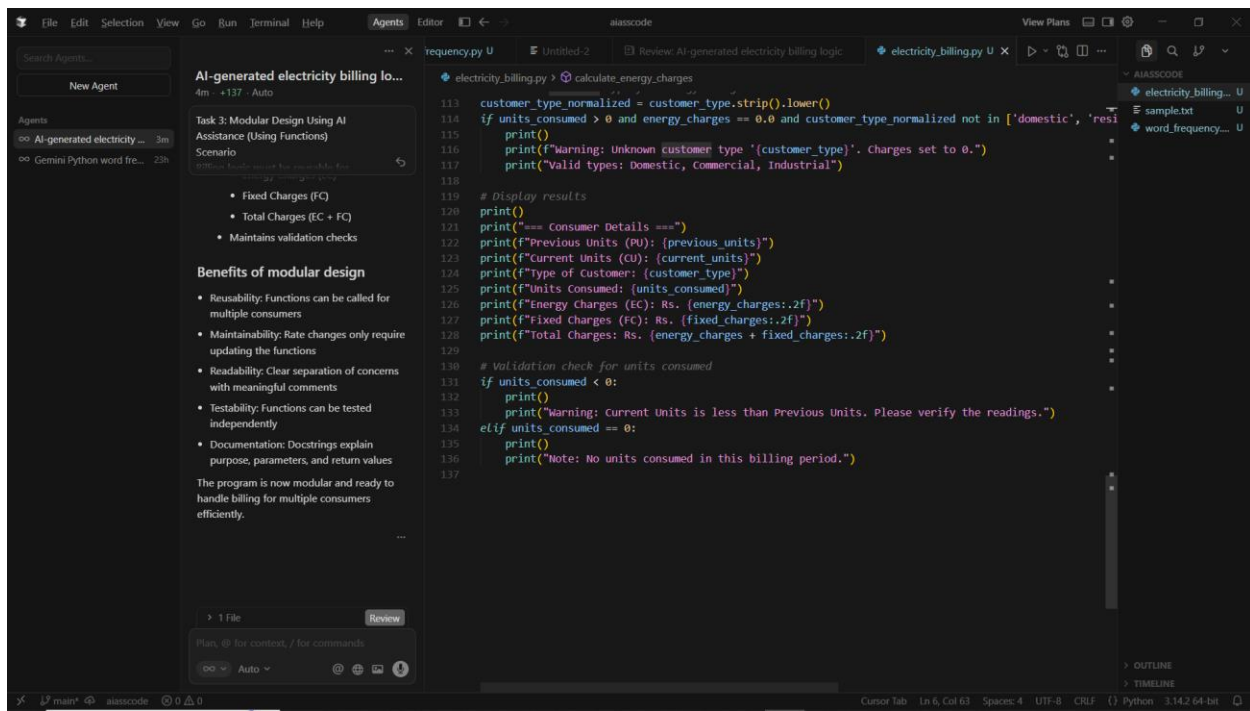
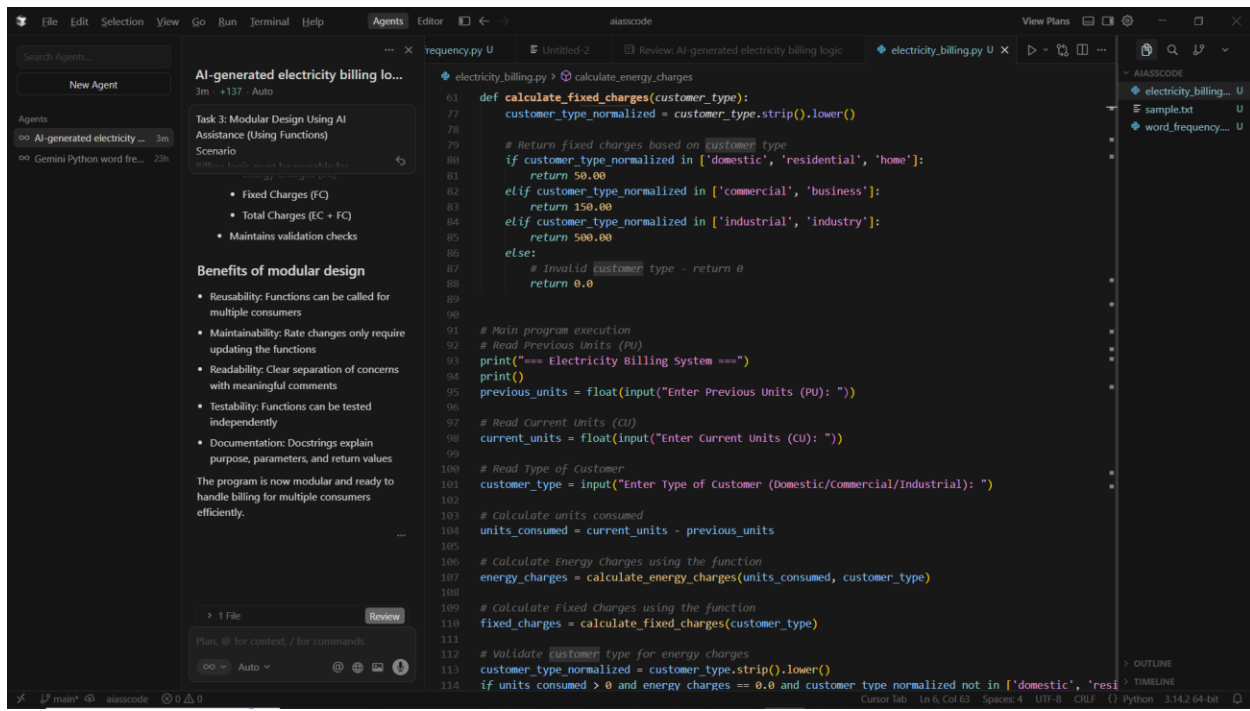
- electricity_billing... U
- sample.txt U
- word_frequency... U

OUTLINE

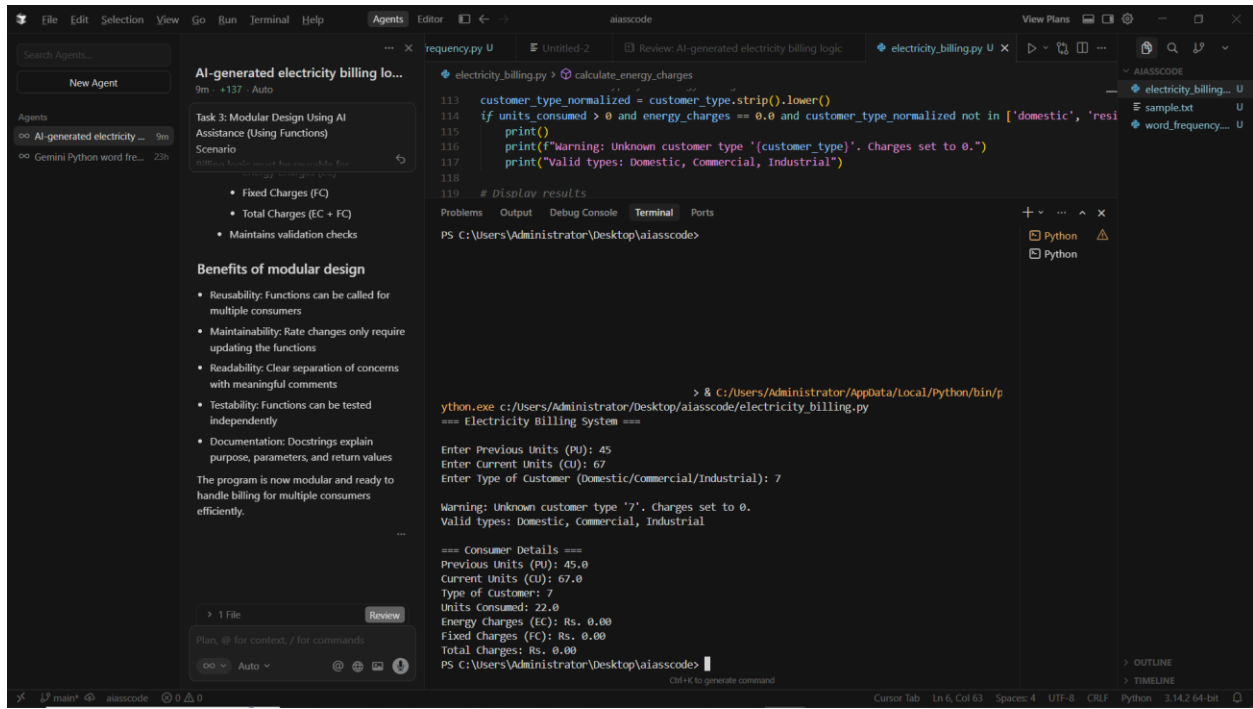
TIMELINE

```
40 def calculate_energy_charges(units_consumed, customer_type):
41     if units_consumed <= 200:
42         return units_consumed * 5.00
43     elif units_consumed <= 500:
44         return 1000.0 + (units_consumed - 200) * 6.00
45     else:
46         return 2800.0 + (units_consumed - 500) * 7.00
47
48     # Calculate energy charges for Industrial customers
49     elif customer_type_normalized in ['industrial', 'industry']:
50         if units_consumed <= 500:
51             return units_consumed * 6.50
52         elif units_consumed <= 1000:
53             return 3250.0 + (units_consumed - 500) * 7.50
54         else:
55             return 7000.0 + (units_consumed - 1000) * 8.50
56
57     # Invalid customer type - return 0
58     else:
59         return 0.0
60
61 def calculate_fixed_charges(customer_type):
62     """
63     Calculate Fixed Charges (FC) based on customer type.
64
65     Fixed charges are monthly charges that don't depend on consumption:
66     - Domestic: Rs. 50.00 per month
67     - Commercial: Rs. 150.00 per month
68     - Industrial: Rs. 500.00 per month
69
70     Args:
71         customer_type (str): Type of customer (Domestic/Commercial/Industrial)
72
73     Returns:
74         float: Fixed charges in rupees
75     """
76     # Normalize customer type for case-insensitive comparison
77     customer_type_normalized = customer_type.strip().lower()
```

Cursor Tab | Ln 6, Col 63 | Spaces: 4 | UTF-8 | CRLF | Python | 3.14.2-64-bit



OUTPUT:



```
electricity_billing.py > calculate_energy_charges
113 customer_type_normalized = customer_type.strip().lower()
114 if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'resi
115 print()
116 print(f"Warning: Unknown customer type '{customer_type}'. Charges set to 0.")
117 print("Valid types: Domestic, Commercial, Industrial")
118
119 # Display results

PS C:\Users\Administrator\Desktop\aiasscode>

> & C:/Users/Administrator/Appdata/Local/Python/bin/py
ython.exe c:/Users/Administrator/Desktop/aiasscode/electricity_billing.py
=== Electricity Billing System ===

Enter Previous Units (PU): 45
Enter Current Units (CU): 67
Enter Type of Customer (Domestic/Commercial/Industrial): 7

Warning: Unknown customer type '7'. Charges set to 0.
Valid types: Domestic, Commercial, Industrial

=== Consumer Details ===
Previous Units (PU): 45.0
Current Units (CU): 67.0
Type of Customer: 7
Units Consumed: 22.0
Energy Charges (EC): Rs. 0.00
Fixed Charges (FC): Rs. 0.00
Total Charges: Rs. 0.00
PS C:\Users\Administrator\Desktop\aiasscode>
```

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

Task Description

Extend the program to calculate:

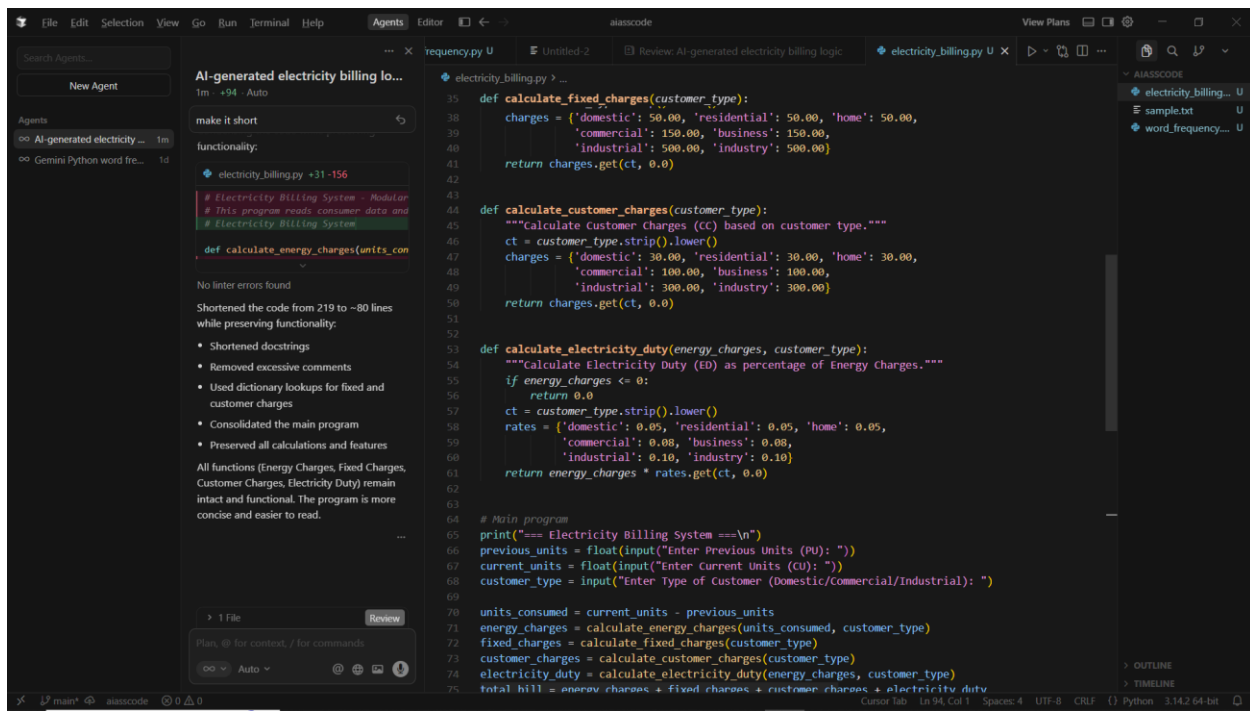
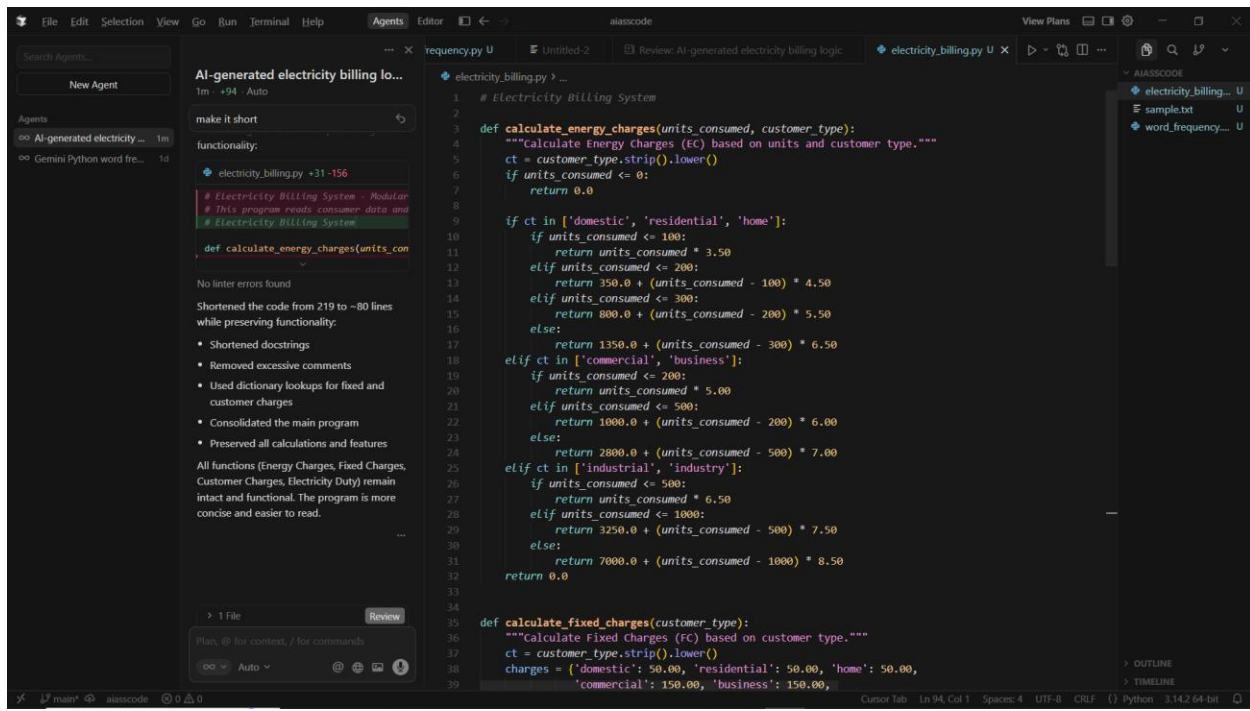
- FC – Fixed Charges
- CC – Customer Charges
- ED – Electricity Duty (percentage of EC)

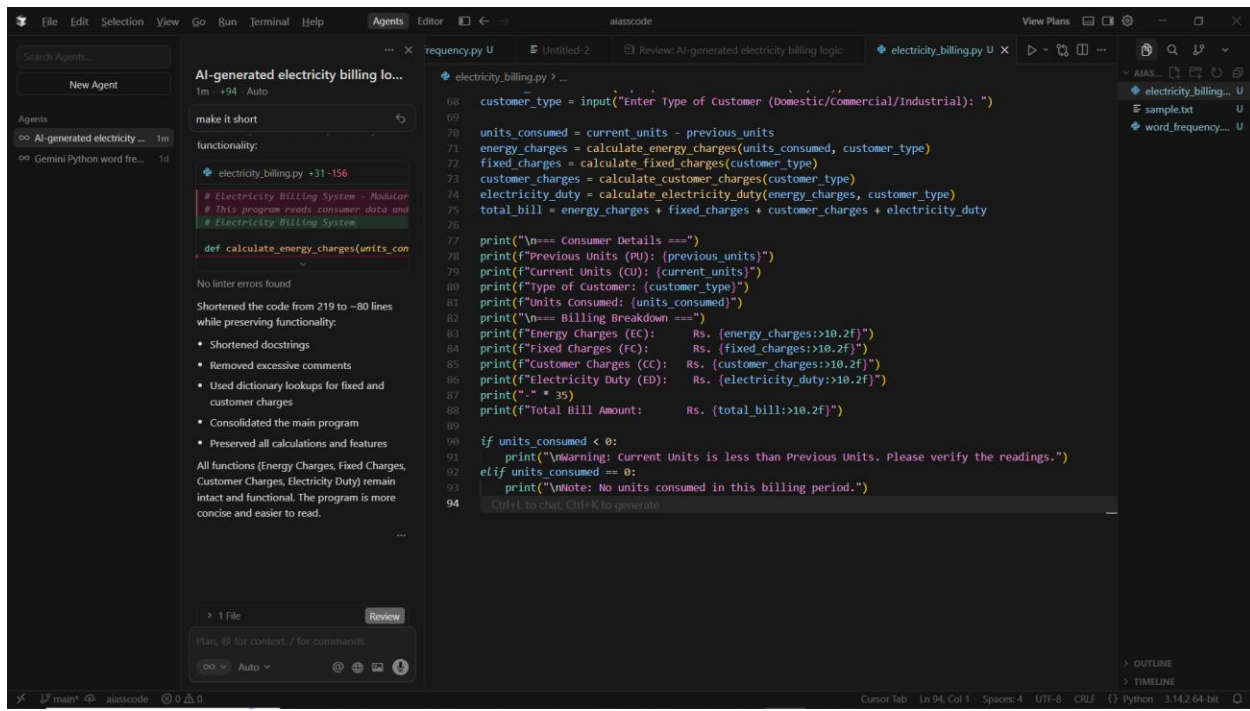
Use AI prompts like:

- “Add electricity duty calculation”
- “Improve billing accuracy”

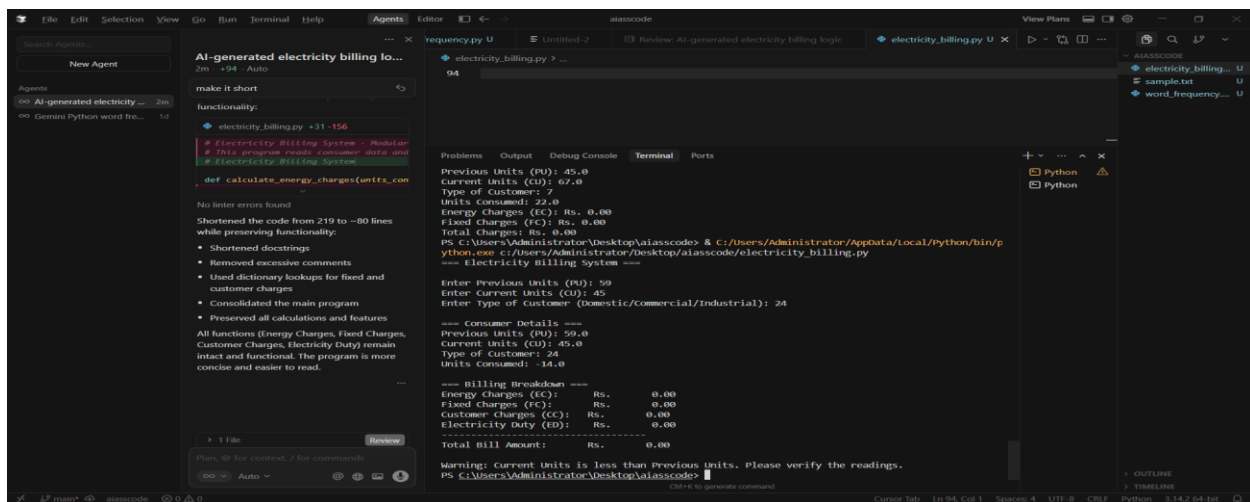
PROMPT:

Write a program to calculate and display an electricity bill using meter readings, customer type, and applicable charges





OUTPUT:



EXPLANATION:

This program calculates an electricity bill by reading previous and current meter readings and the customer type. It computes units consumed, applies slab-based energy charges, adds fixed charges, customer charges, and electricity duty, then displays a detailed bill. It also checks for invalid or zero unit consumption.

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

Task Description

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)

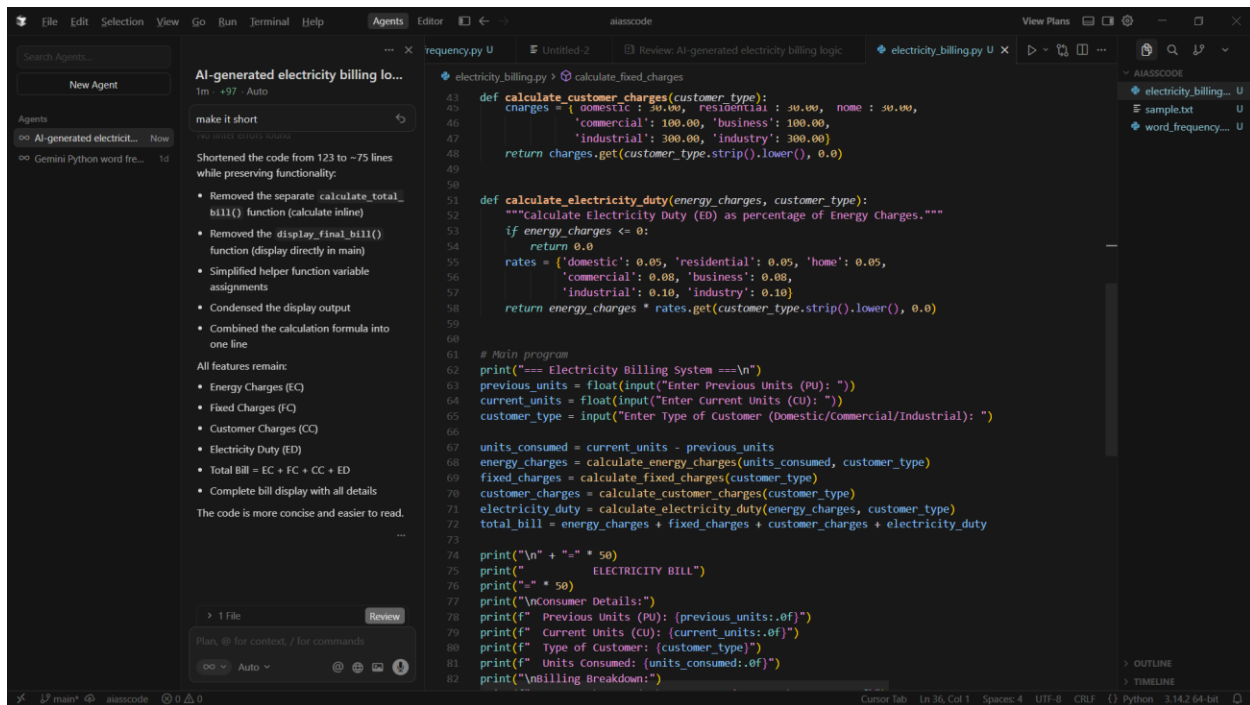
o Electricity Duty (ED)

o Total Bill Amount

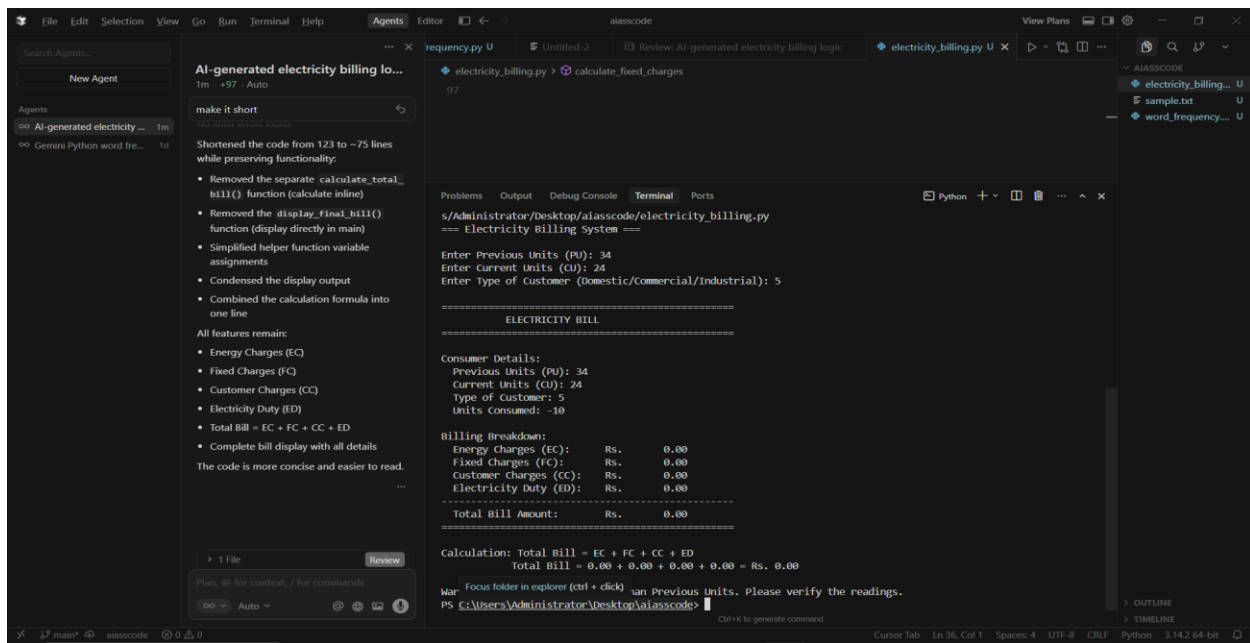
PROMPT:

Write a Python program to calculate and display an electricity bill using meter readings, customer type, and applicable charges.

```
1 # Electricity Billing System
2
3 def calculate_energy_charges(units_consumed, customer_type):
4     """Calculate Energy Charges (EC) based on units and customer type."""
5     ct = customer_type.strip().lower()
6     if units_consumed <= 0:
7         return 0.0
8
9     if ct in ['domestic', 'residential', 'home']:
10         if units_consumed <= 100:
11             return units_consumed * 3.50
12         elif units_consumed <= 200:
13             return 350.0 + (units_consumed - 100) * 4.50
14         elif units_consumed <= 300:
15             return 800.0 + (units_consumed - 200) * 5.50
16         else:
17             return 1350.0 + (units_consumed - 300) * 6.50
18     elif ct in ['commercial', 'business']:
19         if units_consumed <= 200:
20             return units_consumed * 5.00
21         elif units_consumed <= 500:
22             return 1000.0 + (units_consumed - 200) * 6.00
23         else:
24             return 2800.0 + (units_consumed - 500) * 7.00
25     elif ct in ['industrial', 'industry']:
26         if units_consumed <= 500:
27             return units_consumed * 6.50
28         elif units_consumed <= 1000:
29             return 3250.0 + (units_consumed - 500) * 7.50
30         else:
31             return 7000.0 + (units_consumed - 1000) * 8.50
32     return 0.0
33
34 def calculate_fixed_charges(customer_type):
35     """Calculate Fixed Charges (FC) based on customer type."""
36     charges = {'domestic': 50.00, 'residential': 50.00, 'home': 50.00,
37               'commercial': 150.00, 'business': 150.00,
38               'industrial': 500.00, 'industry': 500.00}
```



OUTPUT



EXPLANATION:

This program calculates an electricity bill by reading meter readings and customer type, computing units consumed, applying slab-based charges, adding other charges, and displaying the total bill with basic validation