

ASSINMENT 3.3

NAME:K.Naga Chaitanya

HALLTICKET:2303A51292

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

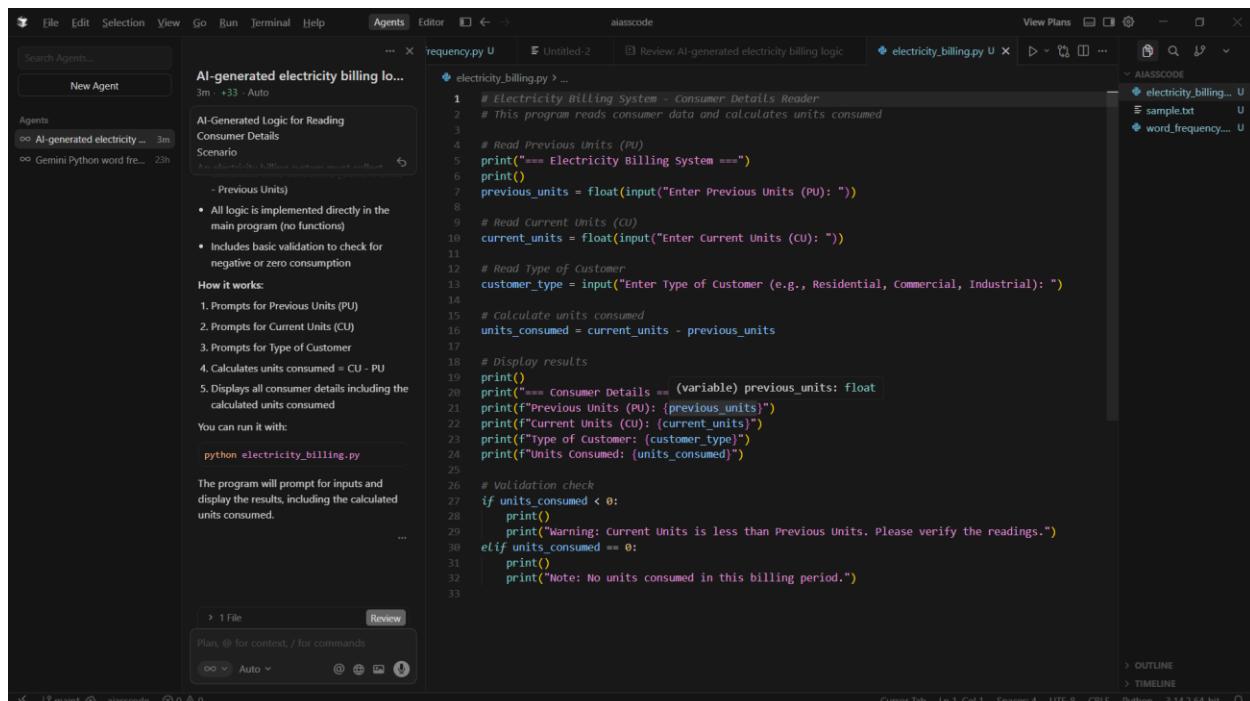
An electricity billing system must collect accurate consumer data.

Task Description

Use an AI tool (GitHub Copilot / Gemini) to generate a Python program that:

- Reads:
 - Previous Units (PU)
 - Current Units (CU)
 - 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 shows the GitHub Copilot interface with the following details:

- File Explorer:** Shows 'Agents' and 'AI-generated electricity...' under 'Agents'.
- Code Editor:** Displays the generated Python code for 'electricity_billing.py'. The code reads consumer data and calculates units consumed.
- Terminal:** Shows the command 'python electricity_billing.py'.
- Output:** Shows the execution of the program, displaying consumer details and validation messages.
- Status Bar:** Shows 'Cursor Tab: In 1, Col 1' and 'Python: 3.14.2 64-bit'.

```
# Electricity Billing System - Consumer Details Reader
# This program reads consumer data and calculates units consumed

# Read Previous Units (PU)
print("== Electricity Billing System ==")
print()
previous_units = float(input("Enter Previous Units (PU): "))

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

# Read Type of Customer
customer_type = input("Enter Type of Customer (e.g., Residential, Commercial, Industrial): ")

# Calculate units consumed
units_consumed = current_units - previous_units

# Display results
print()
print("== Consumer Details ==")
print(f"Previous Units (PU): {previous_units}")
print(f"Current Units (CU): {current_units}")
print(f"Type of Customer: {customer_type}")
print(f"Units Consumed: {units_consumed}")

# Validation check
if units_consumed < 0:
    print()
    print("Warning: Current Units is less than Previous Units. Please verify the readings.")
elif units_consumed == 0:
    print()
    print("Note: No units consumed in this billing period.")
```

OUTPUT:

The screenshot shows a code editor interface with several tabs and panels. On the left, there's a sidebar titled 'Agents' with a section for 'AI-generated electricity ...'. The main area displays Python code for an electricity billing system. The code reads consumer details and calculates units consumed by subtracting previous units from current units. It includes validation for negative or zero consumption and handles errors if current units are less than previous units. The terminal panel at the bottom shows the command 'python electricity_billing.py' being run and its output, which includes user prompts for previous and current units and a warning message.

```
# Electricity Billing System - Consumer Details Reader
# This program reads consumer data and calculates units consumed
# Read Previous Units (PU)
# Read Current Units (CU)
# 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 :\\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:**
 - **Domestic**
 - **Commercial**
 - **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.

```

# Electricity Billing System - Consumer Details Reader
# This program reads consumer data and calculates units consumed

# Read Previous Units (PU)
print("Electricity Billing System")
print("Enter Previous Units (PU): ")
previous_units = float(input())

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

# Read Type of Customer
customer_type = input("Enter Type of Customer (e.g., Residential, Commercial, Industrial): ")
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")

# Calculate units consumed
units_consumed = current_units - previous_units

# Normalize customer type for comparison (case-insensitive)
customer_type_normalized = customer_type.strip().lower()

# Initialize energy charges
energy_charges = 0.0

# Calculate Energy charges (EC) based on customer type and units consumed
# Optimized tiered rate calculation using simplified conditional logic
if units_consumed > 0:
    # Domestic customer rates: 0-100@3.50, 101-200@4.50, 201-300@5.50, 300+@6.50
    if customer_type_normalized in ['domestic', 'residential', 'home']:
        if units_consumed <= 100:
            energy_charges = units_consumed * 3.50
        elif units_consumed <= 200:
            energy_charges = 350.0 + (units_consumed - 100) * 4.50
        elif units_consumed <= 300:
            energy_charges = 800.0 + (units_consumed - 200) * 5.50
        else:
            energy_charges = 1350.0 + (units_consumed - 300) * 6.50
    # Commercial customer rates: 0-200@5.00, 201-500@6.00, 500+@7.00
    elif 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
    # Industrial customer rates: 0-500@6.50, 501-1000@7.50, 1000+@8.50
    elif customer_type_normalized in ['industrial', 'industry']:
        if units_consumed <= 500:
            energy_charges = units_consumed * 6.50
        elif units_consumed <= 1000:
            energy_charges = 3250.0 + (units_consumed - 500) * 7.50
        else:
            energy_charges = 7000.0 + (units_consumed - 1000) * 8.50
    # Invalid customer type
    else:
        print("Warning: Unknown customer type '{customer_type}'. Energy charges set to 0.")
        print("Valid types: Domestic, Commercial, Industrial")

# Display results
print("Consumer Details")
print(f"Previous Units (PU): {previous_units}")
print(f"Current Units (CU): {current_units}")
print(f"Type of Customer: {customer_type}")
print(f"Units Consumed: {units_consumed}")
print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")

# Validation check
if units_consumed < 0:
    print("Warning: Current Units is less than Previous Units. Please verify the readings.")
elif units_consumed == 0:
    print("Note: No units consumed in this billing period.")

```

```

# Electricity Billing System - Consumer Details Reader
# This program reads consumer data and calculates units consumed

# Read Previous Units (PU)
print("Electricity Billing System")
print("Enter Previous Units (PU): ")
previous_units = float(input())

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

# Read Type of Customer
customer_type = input("Enter Type of Customer (e.g., Residential, Commercial, Industrial): ")
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")

# Calculate units consumed
units_consumed = current_units - previous_units

# Normalize customer type for comparison (case-insensitive)
customer_type_normalized = customer_type.strip().lower()

# Initialize energy charges
energy_charges = 0.0

# Calculate Energy charges (EC) based on customer type and units consumed
# Optimized tiered rate calculation using simplified conditional logic
if units_consumed > 0:
    # Domestic customer rates: 0-100@3.50, 101-200@4.50, 201-300@5.50, 300+@6.50
    if customer_type_normalized in ['domestic', 'residential', 'home']:
        if units_consumed <= 100:
            energy_charges = units_consumed * 3.50
        elif units_consumed <= 200:
            energy_charges = 350.0 + (units_consumed - 100) * 4.50
        elif units_consumed <= 300:
            energy_charges = 800.0 + (units_consumed - 200) * 5.50
        else:
            energy_charges = 1350.0 + (units_consumed - 300) * 6.50
    # Commercial customer rates: 0-200@5.00, 201-500@6.00, 500+@7.00
    elif 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
    # Industrial customer rates: 0-500@6.50, 501-1000@7.50, 1000+@8.50
    elif customer_type_normalized in ['industrial', 'industry']:
        if units_consumed <= 500:
            energy_charges = units_consumed * 6.50
        elif units_consumed <= 1000:
            energy_charges = 3250.0 + (units_consumed - 500) * 7.50
        else:
            energy_charges = 7000.0 + (units_consumed - 1000) * 8.50
    # Invalid customer type
    else:
        print("Warning: Unknown customer type '{customer_type}'. Energy charges set to 0.")
        print("Valid types: Domestic, Commercial, Industrial")

# Display results
print("Consumer Details")
print(f"Previous Units (PU): {previous_units}")
print(f"Current Units (CU): {current_units}")
print(f"Type of Customer: {customer_type}")
print(f"Units Consumed: {units_consumed}")
print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")

# Validation check
if units_consumed < 0:
    print("Warning: Current Units is less than Previous Units. Please verify the readings.")
elif units_consumed == 0:
    print("Note: No units consumed in this billing period.")

```

OUTPUT:

```
electricity.billing.py > ...
if 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
...
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>
```

EXPLANATION:This program reads previous and current electricity meter readings and the customer type. It calculates the units consumed, applies slab-based energy charges based on the customer type (Domestic, Commercial, or Industrial), displays the bill details, and checks for invalid or zero readings.

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

Scenario

Billing logic must be reusable for multiple consumers.

Task Description

Use AI assistance to generate a Python program that:

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

AI-generated electricity billing logic

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

Scenario: *Bill multiple consumers based on their consumption levels.*

- 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  # 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
21     # Normalize customer type for case-insensitive comparison
22     customer_type_normalized = customer_type.strip().lower()
23
24
25     # Return 0 if no units consumed or invalid input
26     if units_consumed <= 0:
27         return 0.0
28
29     # calculate energy charges for domestic customers
30     if customer_type_normalized in ['domestic', 'residential', 'home']:
31         if units_consumed <= 100:
32             return units_consumed * 3.50
33         elif units_consumed <= 200:
34             return 350.0 + (units_consumed - 100) * 4.50
35         elif units_consumed <= 300:
36             return 800.0 + (units_consumed - 200) * 5.50
37         else:
38             return 1350.0 + (units_consumed - 300) * 6.50
39
40     # calculate energy charges for commercial customers
41     elif customer_type_normalized in ['commercial', 'business']:
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

```

AI-generated electricity billing logic

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

Scenario: *Bill multiple consumers based on their consumption levels.*

- 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  # 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          If units_consumed <= 200:
9              return units_consumed * 5.00
10         elif units_consumed <= 500:
11             return 1000.0 + (units_consumed - 200) * 6.00
12         else:
13             return 2800.0 + (units_consumed - 500) * 7.00
14
15     # calculate energy charges for industrial customers
16     elif customer_type_normalized in ['industrial', 'industry']:
17         if units_consumed <= 500:
18             return units_consumed * 6.50
19         elif units_consumed <= 1000:
20             return 3250.0 + (units_consumed - 500) * 7.50
21         else:
22             return 7000.0 + (units_consumed - 1000) * 8.50
23
24     # Invalid customer type - return 0
25     else:
26         return 0.0
27
28
29
30     def calculate_fixed_charges(customer_type):
31         """
32             Calculate Fixed Charges (FC) based on customer type.
33
34             Fixed charges are monthly charges that don't depend on consumption:
35             - Domestic: Rs. 50.00 per month
36             - Commercial: Rs. 150.00 per month
37             - Industrial: Rs. 500.00 per month
38
39         Args:
40             customer_type (str): Type of customer (Domestic/Commercial/Industrial)
41
42         Returns:
43             float: Fixed charges in rupees
44         """
45
46         # Normalize customer type for case-insensitive comparison
47         customer_type_normalized = customer_type.strip().lower()
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77

```

AI-generated electricity billing logic

3m · +137 · Auto

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

Scenario

Electricity billing system for multiple consumers.

- 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 def calculate_fixed_charges(customer_type):
2     customer_type_normalized = customer_type.strip().lower()
3
4     # Return fixed charges based on customer type
5     if customer_type_normalized in ['domestic', 'residential', 'home']:
6         return 50.00
7     elif customer_type_normalized in ['commercial', 'business']:
8         return 150.00
9     elif customer_type_normalized in ['industrial', 'industry']:
10        return 500.00
11    else:
12        # Invalid customer type - return 0
13        return 0.0
14
15
16    # Main program execution
17    # Read Previous Units (PU)
18    print("==> Electricity Billing System ==>")
19    print()
20    previous_units = float(input("Enter Previous Units (PU): "))
21
22    # Read Current Units (CU)
23    current_units = float(input("Enter Current Units (CU): "))
24
25    # Read Type of Customer
26    customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")
27
28    # Calculate units consumed
29    units_consumed = current_units - previous_units
30
31    # Calculate Energy Charges using the function
32    energy_charges = calculate_energy_charges(units_consumed, customer_type)
33
34    # calculate Fixed charges using the function
35    fixed_charges = calculate_fixed_charges(customer_type)
36
37    # Validate customer type for energy charges
38    customer_type_normalized = customer_type.strip().lower()
39    if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'residential', 'home']:
40        print("Warning: Unknown customer type '{customer_type}'. Charges set to 0.0")
41
42    # Display results
43    print()
44    print("==> Consumer Details ==>")
45    print(f"Previous Units (PU): {previous_units}")
46    print(f"Current Units (CU): {current_units}")
47    print(f"Type of customer: {customer_type}")
48    print(f"Units Consumed: {units_consumed}")
49    print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")
50    print(f"Fixed Charges (FC): Rs. {fixed_charges:.2f}")
51    print(f"Total Charges: Rs. {energy_charges + fixed_charges:.2f}")
52
53    # Validation check for units consumed
54    if units_consumed < 0:
55        print()
56        print("Warning: Current Units is less than Previous Units. Please verify the readings.")
57    elif units_consumed == 0:
58        print()
59        print("Note: No units consumed in this billing period.")

```

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

AI-generated electricity billing logic

4m · +137 · Auto

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

Scenario

Electricity billing system for multiple consumers.

- 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 customer_type_normalized = customer_type.strip().lower()
2 if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'residential', 'home']:
3     print()
4     print("Warning: Unknown customer type '{customer_type}'. Charges set to 0.0")
5     print("Valid types: domestic, commercial, industrial")
6
7
8    # Display results
9    print()
10   print("==> Consumer Details ==>")
11   print(f"Previous Units (PU): {previous_units}")
12   print(f"Current Units (CU): {current_units}")
13   print(f"Type of customer: {customer_type}")
14   print(f"Units Consumed: {units_consumed}")
15   print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")
16   print(f"Fixed Charges (FC): Rs. {fixed_charges:.2f}")
17   print(f"Total Charges: Rs. {energy_charges + fixed_charges:.2f}")
18
19    # Validation check for units consumed
20    if units_consumed < 0:
21        print()
22        print("Warning: Current Units is less than Previous Units. Please verify the readings.")
23    elif units_consumed == 0:
24        print()
25        print("Note: No units consumed in this billing period.")

```

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

OUTPUT:

The screenshot shows the AISSCODE interface with the following details:

- Left Panel:** Shows a tree view of "Agents" with "AI-generated electricity..." selected.
- Middle Panel:** A modal titled "AI-generated electricity billing lo..." displays:
 - Task 3: Modular Design Using AI Assistance (Using Functions)**
 - Scenario:** "Differentiator: modular design using functions"
 - 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.
- Right Panel:** The "electricity_billing.py" code editor shows:

```
customer_type_normalized = customer_type.strip().lower()
if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'residential']:
    print("Warning: Unknown customer type '{customer_type}'. charges set to 0.")
    print("Valid types: Domestic, Commercial, Industrial")
```
- Terminal:** The terminal window shows the execution of the script and its output:

```
> & 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): 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
```

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

AI-generated electricity billing logic

```

1m +94 Auto
make it short
functionality:
electricity_billing.py +31-156
# Electricity Billing System - Modular
# This program reads consumer data and
# Electricity Billing System

def calculate_energy_charges(units_consumed):
    """Calculate Energy Charges (EC) based on units and customer type."""
    ct = customer_type.strip().lower()
    if units_consumed <= 0:
        return 0.0
    if ct in ['domestic', 'residential', 'home']:
        if units_consumed <= 100:
            return units_consumed * 3.50
        elif units_consumed <= 200:
            return 350.0 + (units_consumed - 100) * 4.50
        elif units_consumed <= 300:
            return 800.0 + (units_consumed - 200) * 5.50
        else:
            return 1350.0 + (units_consumed - 300) * 6.50
    elif ct in ['commercial', 'business']:
        if units_consumed <= 200:
            return units_consumed * 5.00
        elif units_consumed <= 500:
            return 1000.0 + (units_consumed - 200) * 6.00
        else:
            return 2800.0 + (units_consumed - 500) * 7.00
    elif ct in ['industrial', 'industry']:
        if units_consumed <= 500:
            return units_consumed * 6.50
        elif units_consumed <= 1000:
            return 3250.0 + (units_consumed - 500) * 7.50
        else:
            return 7000.0 + (units_consumed - 1000) * 8.50
    return 0.0

def calculate_fixed_charges(customer_type):
    """Calculate Fixed Charges (FC) based on customer type."""
    ct = customer_type.strip().lower()
    charges = {'domestic': 50.00, 'residential': 50.00, 'home': 50.00,
               'commercial': 150.00, 'business': 150.00,
               'industrial': 500.00, 'industry': 500.00}
    return charges.get(ct, 0.0)

def calculate_customer_charges(customer_type):
    """Calculate Customer Charges (CC) based on customer type."""
    ct = customer_type.strip().lower()
    charges = {'domestic': 30.00, 'residential': 30.00, 'home': 30.00,
               'commercial': 100.00, 'business': 100.00,
               'industrial': 300.00, 'industry': 300.00}
    return charges.get(ct, 0.0)

def calculate_electricity_duty(energy_charges, customer_type):
    """Calculate Electricity Duty (ED) as percentage of Energy Charges."""
    if energy_charges <= 0:
        return 0.0
    ct = customer_type.strip().lower()
    rates = {'domestic': 0.05, 'residential': 0.05, 'home': 0.05,
             'commercial': 0.08, 'business': 0.08,
             'industrial': 0.10, 'industry': 0.10}
    return energy_charges * rates.get(ct, 0.0)

# Main program
print("== Electricity Billing System ==\n")
previous_units = float(input("Enter Previous Units (PU): "))
current_units = float(input("Enter Current Units (CU): "))
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")

units_consumed = current_units - previous_units
energy_charges = calculate_energy_charges(units_consumed, customer_type)
fixed_charges = calculate_fixed_charges(customer_type)
customer_charges = calculate_customer_charges(customer_type)
electricity_duty = calculate_electricity_duty(energy_charges, customer_type)
total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

```

AI-generated electricity billing logic

```

1m +94 Auto
make it short
functionality:
electricity_billing.py +31-156
# Electricity Billing System - Modular
# This program reads consumer data and
# Electricity Billing System

def calculate_energy_charges(units_consumed):
    """Calculate Energy Charges (EC) based on units and customer type."""
    ct = customer_type.strip().lower()
    if units_consumed <= 0:
        return 0.0
    if ct in ['domestic', 'residential', 'home']:
        if units_consumed <= 100:
            return units_consumed * 3.50
        elif units_consumed <= 200:
            return 350.0 + (units_consumed - 100) * 4.50
        elif units_consumed <= 300:
            return 800.0 + (units_consumed - 200) * 5.50
        else:
            return 1350.0 + (units_consumed - 300) * 6.50
    elif ct in ['commercial', 'business']:
        if units_consumed <= 200:
            return units_consumed * 5.00
        elif units_consumed <= 500:
            return 1000.0 + (units_consumed - 200) * 6.00
        else:
            return 2800.0 + (units_consumed - 500) * 7.00
    elif ct in ['industrial', 'industry']:
        if units_consumed <= 500:
            return units_consumed * 6.50
        elif units_consumed <= 1000:
            return 3250.0 + (units_consumed - 500) * 7.50
        else:
            return 7000.0 + (units_consumed - 1000) * 8.50
    return 0.0

def calculate_fixed_charges(customer_type):
    """Calculate Fixed Charges (FC) based on customer type."""
    ct = customer_type.strip().lower()
    charges = {'domestic': 50.00, 'residential': 50.00, 'home': 50.00,
               'commercial': 150.00, 'business': 150.00,
               'industrial': 500.00, 'industry': 500.00}
    return charges.get(ct, 0.0)

def calculate_customer_charges(customer_type):
    """Calculate Customer Charges (CC) based on customer type."""
    ct = customer_type.strip().lower()
    charges = {'domestic': 30.00, 'residential': 30.00, 'home': 30.00,
               'commercial': 100.00, 'business': 100.00,
               'industrial': 300.00, 'industry': 300.00}
    return charges.get(ct, 0.0)

def calculate_electricity_duty(energy_charges, customer_type):
    """Calculate Electricity Duty (ED) as percentage of Energy Charges."""
    if energy_charges <= 0:
        return 0.0
    ct = customer_type.strip().lower()
    rates = {'domestic': 0.05, 'residential': 0.05, 'home': 0.05,
             'commercial': 0.08, 'business': 0.08,
             'industrial': 0.10, 'industry': 0.10}
    return energy_charges * rates.get(ct, 0.0)

# Main program
print("== Electricity Billing System ==\n")
previous_units = float(input("Enter Previous Units (PU): "))
current_units = float(input("Enter Current Units (CU): "))
customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")

units_consumed = current_units - previous_units
energy_charges = calculate_energy_charges(units_consumed, customer_type)
fixed_charges = calculate_fixed_charges(customer_type)
customer_charges = calculate_customer_charges(customer_type)
electricity_duty = calculate_electricity_duty(energy_charges, customer_type)
total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

```

The screenshot shows the AISSCODE IDE interface. The main window displays the Python script `electricity_billng.py` with the following code:

```

# Electricity Billing System - Modular
# This program reads consumer data and
# Electricity Billing System

def calculate_energy_charges(units_consumed):
    # Slab-based energy charges
    if units_consumed < 0:
        print("\nWarning: Current Units is less than Previous Units. Please verify the readings.")
    elif units_consumed == 0:
        print("\nNote: No units consumed in this billing period.")

    # Calculate total bill amount
    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
    print(f"\nTotal Bill Amount: Rs. {total_bill:.2f}")

    # Print billing breakdown
    print("==== Consumer Details ===")
    print(f"Previous Units (PU): {previous_units}")
    print(f"Current Units (CU): {current_units}")
    print(f"Type of Customer: {customer_type}")
    print(f"Units Consumed: {units_consumed}")
    print("==== Billing Breakdown ===")
    print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")
    print(f"Fixed Charges (FC): Rs. {fixed_charges:.2f}")
    print(f"Customer Charges (CC): Rs. {customer_charges:.2f}")
    print(f"Electricity Duty (ED): Rs. {electricity_duty:.2f}")
    print(f"\nTotal * 35")
    print(f"\nTotal Bill Amount: Rs. {total_bill:.2f}")

if __name__ == "__main__":
    main()

```

The code is annotated with AI-generated comments and documentation. The interface includes tabs for `frequency.py`, `Untitled-2`, and `Review: AI-generated electricity billing logic`. The status bar at the bottom shows the current tab is `frequency.py`.

OUTPUT:

The screenshot shows the AISSCODE IDE interface with the terminal tab active, displaying the execution output of the `electricity_billng.py` script:

```

PS C:\Users\Administrator\Desktop\aiasscode> & c:/Users/Administrator/appData/local/Python/bin/python.exe c:/Users/Administrator/Desktop/aiasscode/electricity_billng.py
==== Electricity Billing System ===

Enter Previous Units (PU): 45.0
Enter Current Units (CU): 45
Enter Type of customer (Domestic/Commercial/Industrial): 24
==== Consumer Details ===
Previous Units (PU): 45.0
Current Units (CU): 45
Type of Customer: 24
Units Consumed: 0.0
==== Billing Breakdown ===
Energy Charges (EC): Rs. 0.00
Fixed Charges (FC): Rs. 0.00
Customer Charges (CC): Rs. 0.00
Electricity Duty (ED): Rs. 0.00
Total Bill Amount: Rs. 0.00

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

```

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.

The screenshot shows the AISSCODE IDE interface with the electricity_billing.py file open. The code implements a billing logic based on meter readings and customer type. It includes functions for calculating energy charges, fixed charges, and customer charges, and a main function to calculate the total bill. The code is annotated with comments explaining the logic for different customer types and consumption levels.

```
AI-generated electricity billing logic
make it short
Shortened the code from 123 to ~75 lines while preserving functionality:
• Removed the separate calculate_total_bill() function (calculate inline)
• Removed the display_final_bill() function (display directly in main)
• Simplified helper function variable assignments
• Condensed the display output
• Combined the calculation formula into one line
All features remain:
• Energy Charges (EC)
• Fixed Charges (FC)
• Customer Charges (CC)
• Electricity Duty (ED)
• Total Bill = EC + FC + CC + ED
• Complete bill display with all details
The code is more concise and easier to read.

def calculate_energy_charges(units_consumed, customer_type):
    """
    Calculate Energy Charges (EC) based on units and customer type.
    """
    ct = customer_type.strip().lower()
    if units_consumed <= 0:
        return 0.0
    if ct in ['domestic', 'residential', 'home']:
        if units_consumed < 100:
            return units_consumed * 3.50
        elif units_consumed <= 200:
            return 350.0 + (units_consumed - 100) * 4.50
        elif units_consumed <= 300:
            return 800.0 + (units_consumed - 200) * 5.50
        else:
            return 1350.0 + (units_consumed - 300) * 6.50
    elif ct in ['commercial', 'business']:
        if units_consumed < 200:
            return units_consumed * 5.00
        elif units_consumed <= 500:
            return 1000.0 + (units_consumed - 200) * 6.00
        else:
            return 2800.0 + (units_consumed - 500) * 7.00
    elif ct in ['industrial', 'industry']:
        if units_consumed < 500:
            return units_consumed * 6.50
        elif units_consumed <= 1000:
            return 3250.0 + (units_consumed - 500) * 7.50
        else:
            return 7000.0 + (units_consumed - 1000) * 8.50
    return 0.0

def calculate_fixed_charges(customer_type):
    """
    Calculate Fixed Charges (FC) based on customer type.
    """
    charges = {'domestic': 50.00, 'residential': 50.00, 'home': 50.00,
              'commercial': 150.00, 'business': 150.00,
              'industrial': 500.00, 'industry': 500.00}
```

The screenshot shows a code editor with multiple tabs open. The main tab contains Python code for an electricity billing system. The code includes functions for calculating fixed charges, customer charges based on type, and electricity duty. It also includes a main program loop for taking user input and printing a bill summary.

```
File Edit Selection View Go Run Terminal Help Agents Editor aiascode Review Plans ...
```

```
Search Agents... New Agent
```

```
Agents
```

```
AI-generated electricity... Now
```

```
Gemini Python word fre... 1d
```

```
AI-generated electricity billing lo... 1m + 97 - Auto
```

```
make it short
```

```
Shortened the code from 123 to ~75 lines while preserving functionality:
```

- Removed the separate `calculate_total_bill()` function (calculate inline)
- Removed the `display_final_bill()` function (display directly in main)
- Simplified helper function variable assignments
- Condensed the display output
- Combined the calculation formula into one line

```
All features remain:
```

- Energy Charges (EC)
- Fixed Charges (FC)
- Customer Charges (CC)
- Electricity Duty (ED)
- Total Bill = EC + FC + CC + ED
- Complete bill display with all details

```
The code is more concise and easier to read.
```

```
review.py U Untitled-2 Review: AI-generated electricity billing logic electricity_billing.py U
```

```
electricity_billing.py > calculate_fixed_charges
```

```
43     def calculate_customer_charges(customer_type):
```

```
44         charges = { 'domestic' : 50.00, 'residential' : 50.00, 'home' : 50.00,
```

```
45             'commercial' : 100.00, 'business' : 100.00,
```

```
46             'industrial' : 300.00, 'industry' : 300.00 }
```

```
47         return charges.get(customer_type.strip().lower(), 0.0)
```

```
48 
```

```
49 
```

```
50 
```

```
51 def calculate_electricity_duty(energy_charges, customer_type):
```

```
52     """Calculate Electricity Duty (ED) as percentage of Energy Charges."""
```

```
53     if energy_charges <= 0:
```

```
54         return 0.0
```

```
55     rates = { 'domestic': 0.05, 'residential': 0.05, 'home': 0.05,
```

```
56             'commercial': 0.08, 'business': 0.08,
```

```
57             'industrial': 0.10, 'industry': 0.10 }
```

```
58     return energy_charges * rates.get(customer_type.strip().lower(), 0.0)
```

```
59 
```

```
60 
```

```
61 # Main program
```

```
62 print("== Electricity Billing System ==\n")
```

```
63 previous_units = float(input("Enter Previous Units (PU): "))
```

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

```
65 customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ")
```

```
66 
```

```
67 units_consumed = current_units - previous_units
```

```
68 energy_charges = calculate_energy_charges(units_consumed, customer_type)
```

```
69 fixed_charges = calculate_fixed_charges(customer_type)
```

```
70 customer_charges = calculate_customer_charges(customer_type)
```

```
71 electricity_duty = calculate_electricity_duty(energy_charges, customer_type)
```

```
72 total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
```

```
73 
```

```
74 print("\n" + "=" * 50)
```

```
75 print(" " * 25 + "ELECTRICITY BILL")
```

```
76 print("=" * 50)
```

```
77 print("\nConsumer Details:")
```

```
78 print(f" Previous Units (PU): {previous_units:.0f}")
```

```
79 print(f" Current Units (CU): {current_units:.0f}")
```

```
80 print(f" Type of Customer: {customer_type}")
```

```
81 print(f" Units Consumed: {units_consumed:.0f}")
```

```
82 print("\nBilling Breakdown:")
```

```
Review
```

```
> 1 File Plan, @ for context, / for commands
```

```
main.py aiascode
```

```
Auto @
```

```
OUTLINE
```

```
TIMELINE
```

```
Cursor Tab Ln 36 Col 1 Spaces: 4 UTF-8 CRLF () Python 3.14.2 64-bit
```

OUTPUT

The screenshot shows the AISSCODE interface with several tabs open. The main tab displays AI-generated code for an electricity billing system, which has been simplified from 123 lines to 97 lines. The code uses functions like `calculate_total_bill()` and `display_final_bill()`. It includes a terminal window showing a sample run where the user enters previous and current units, selects a customer type, and sees the bill details. The terminal output is as follows:

```
s/Administrator/Desktop/aiasscode/electricity_billing.py
== Electricity Billing System ==

Enter Previous Units (PU): 34
Enter Current Units (CU): 24
Enter Type of Customer (Domestic/commercial/Industrial): 5

=====
ELECTRICITY BILL
=====

Consumer Details:
Previous Units (PU): 34
Current Units (CU): 24
Type of Customer: 5
Units Consumed: -10

Billing Breakdown:
Energy Charges (EC): Rs. 0.00
Fixed Charges (FC): Rs. 0.00
Customer Charges (CC): Rs. 0.00
Electricity Duty (ED): Rs. 0.00

Total Bill Amount: Rs. 0.00

=====

Calculation: Total Bill = EC + FC + CC + ED
Total Bill = 0.00 + 0.00 + 0.00 + 0.00 = Rs. 0.00
```

At the bottom, a warning message says "Focus folder in explorer (ctrl + click) on Previous Units. Please verify the readings." and "PS C:\Users\Administrator\Desktop\aiasscode".

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