

## **ASSIGNMENT-03**

**NAME:** B.Manishwar

**HALL TICKET:**2303A51276

**BATCH:**05

### **Q) 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)

#### **Expected Output**

- Correct input reading
- Units consumed calculation
- Screenshot showing AI-generated code
- Sample input and output

#### **PROMPT:**

Create a Python program that reads previous units, current units, and customer type, calculates units consumed, and displays the result. Write all logic in the main program without using functions.

#### **CODE:**

```

# Display header
print("#" * 60)
print("Welcome to Electricity Bill Generation System")
print("Task 1: Reading Consumer Details")
print("#" * 60)

# Read Previous Units (PU)
print("Enter Previous Units:")
print("#" * 60)
try:
    previous_units = float(input("Previous Units (PU):"))
except ValueError:
    print("Error: Please enter a valid number for Previous Units.")
    previous_units = 0.0

# Read Current Units (CU)
try:
    current_units = float(input("Current Units (CU):"))
except ValueError:
    print("Error: Please enter a valid number for Current Units.")
    current_units = 0.0

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

# Validate that current units is greater than or equal to previous units
if current_units < previous_units:
    print("\u26a0 Warning: Current Units cannot be less than Previous Units!")
    print("Please verify your input.")

# Calculate units consumed
units_consumed = current_units - previous_units

# Display results
print()
print("#" * 60)
print("CONSUMER DETAILS SUMMARY")
print("Previous Units (PU) : (previous_units:.2f) units")
print("Current Units (CU) : (current_units:.2f) units")
print("Type of Customer : (customer_type)")
print("Units Consumed : (units_consumed:.2f) units")
print("#" * 60)

# Additional validation message
if units_consumed < 0:
    print("\u26a0 Error: Negative units consumed detected!")
    print("Please check your input values.")
elif units_consumed == 0:
    print("\u26a0 Note: No units consumed in this billing period.")
else:
    print("\u2708 Units consumed calculated successfully: (units_consumed:.2f) units")

print("\n" * 60)
print("Task 1 Completed Successfully!")
print("\n" * 60)

```

## OUTPUT:

```

source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
"/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
(base) bodla.manishwar@BodlaManishwars-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
(base) bodla.manishwar@BodlaManishwars-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task1_Consumer_Details.py"
=====
TGNPCL - Electricity Bill Generation System
Task 1: Reading Consumer Details
=====

Enter Consumer Details:
=====
Previous Units (PU): 120
Current Units (CU): 185
Type of Customer (e.g., Domestic, Commercial, Industrial): Domestic
=====

CONSUMER DETAILS SUMMARY
=====
Previous Units (PU) : 120.00 units
Current Units (CU) : 185.00 units
Type of Customer : Domestic
Units Consumed : 65.00 units
=====

Units consumed calculated successfully: 65.00 units
=====

Task 1 Completed Successfully!
=====

(.venv) (base) bodla.manishwar@BodlaManishwars-Laptop AIC %

```

## **CODE EXPLANATION:**

This program collects consumer details such as previous meter units, current meter units, and customer type, then calculates the electricity units consumed by subtracting previous units from current units. It includes basic input validation to handle incorrect values and ensures accurate calculations. The logic is written directly in the main program without using functions, making it simple, clear, and suitable for a lab-based electricity billing application.

**Q) 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:
    - “Simplify energy charge calculation logic”
    - “Optimize conditional statements”

### Expected Output

- Correct EC calculation
  - Clear conditional logic
  - Original and improved versions (optional)
  - Sample execution results

## PROMPT:

Modify the Python billing code to calculate energy charges using conditional statements for Domestic, Commercial, and Industrial consumers and display the result clearly.

## **CODE:**

The screenshot shows a Jupyter Notebook interface with two code cells. The first cell contains the following Python code:

```
# Task 2: Energy Charges Calculation
# This script demonstrates both original and improved versions of the energy charges calculator.

# Task 2: Energy Charges Calculation
# This version has optimized conditions

def calculate_energy_charges(units, customer_type):
    """Calculate energy charges based on units consumed and customer type."""
    if customer_type == "Domestic":
        return units * 0.15
    elif customer_type == "Commercial":
        return units * 0.20
    else:
        return units * 0.18

def main():
    """Main function to run all demonstrations."""
    # Compare both versions
    compare_versions()

    # Show original version
    demonstrate_original_version()

    # Show improved version
    demonstrate_improved_version()

    # Interactive calculator
    print("\n\n" + "="*70)
    print("Would you like to calculate energy charges for custom values?")
    print("="*70)
    response = input("Enter 'y' for interactive calculator, or any other key to exit: ").strip().lower()

    if response == 'y':
        interactive_calculation()

    else:
        print("Exiting the program...")

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

The second cell contains the following Python code:

```
# Task 2: Energy Charges Calculation
# This project demonstrates energy charges calculation logic.

# Task 2: Energy Charges Calculation
# This is the initial AI-generated code

# Task 2: Energy Charges Calculation
# This version has optimized conditions

# Task 2: Energy Charges Calculation
# This script demonstrates both original and improved versions of the energy charges calculator.

# Task 2: Energy Charges Calculation
# This version has optimized conditions

# Task 2: Energy Charges Calculation
# This version has optimized conditions
```

## OUTPUT:

The screenshot shows a code editor with two tabs open: `Task2_Energy_Charges_Main.py` and `Task2_Energy_Charges_Improved.py`. The `Improved` version is annotated with green highlights and text:

- `def main():` is annotated with `Original EC`.
- `if response == 'y':` is annotated with `Improved EC`.
- `interactive_calculation()` is annotated with `Match`.

The `Task2_Energy_Charges_Main.py` file contains the following code:

```
def main():
    if response == 'y':
        interactive_calculation()
    else:
        print("n = "+str(n))
        print("Thank you for using Energy Charges Calculator!!!")
        print("n = "+str(n))

    if __name__ == "__main__":
        main()

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

The `Task2_Energy_Charges_Improved.py` file contains the following code:

```
def main():
    if response == 'y':
        interactive_calculation()
    else:
        print("n = "+str(n))
        print("Thank you for using Energy Charges Calculator!!!")
        print("n = "+str(n))

    if __name__ == "__main__":
        main()

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

Annotations for the improved version highlight the removal of redundant code and the use of more readable variable names.

```
def main():
    if response == 'y':
        interactive_calculation()
    else:
        print("n = "+str(n))
        print("Thank you for using Energy Charges Calculator!")
        print("c = "+str(c))

    if __name__ == "__main__":
        main()

# Task 2: Energy Charges Calculation
# This is the initial AT-generated code
# This version has optimized conditions

# Task 2: Energy Charges Calculation
# This script demonstrates both original and improved versions

# Task 2: Energy Charges Calculation
# This project demonstrates energy charges calculation task.

# Task 2: Energy Charges Calculation
# Task 2: Energy Charges Calculation has optimized conditions
```

Exploring 1 file  
Creating a README for the energy charges calculation task.

File Task2\_Energy\_Charges\_Main.py +190  
File Task2\_Energy\_Charges\_Improved.py +149  
File Task2\_Energy\_Charges\_Original.py +73

Checked-to-do list  
No file errors found

File Plan: @ for context, / for commands  
File Review: x Undo All Keep All  
File Auto: 0

File AIC  
File Review Next File: AIC  
File \_pycache\_\_  
File .env  
File README\_Task2.md  
File README\_Task3.md  
File sample\_data.csv  
File Task2\_Energy\_Charger.py  
File Task2\_Energy\_Charger.ipynb  
File Task2\_Improvement.ipynb  
File Task3\_CSV\_Data.ipynb

File Problems Output Debug Console Terminal Ports  
File Input validation and error handling  
File Clear separation of concerns

Sample Calculations:

Units	Customer Type	Energy Charges	Description
50.0	Domestic	£10.00	Low consumption - Domestic
150.0	Domestic	£57.00	Medium consumption - Domestic
150.0	Commercial	£12.00	Medium consumption - Commercial
350.0	Industrial	£2,280.00	High consumption - Industrial
0.0	Commercial	£0.00	Zero consumption - Commercial
0.0	Domestic	£0.00	Zero consumption - Domestic
500.0	Industrial	£3,480.00	Very high consumption - Industrial

Rate Structure Breakdown Example:  
For 250 units consumed by Commercial customer:  
Tier 0-100 units: 100.0 units x £4.00 = £400.00  
Tier 100-200 units: 100.0 units x £5.00 = £500.00  
Tier 200-300 units: 50.0 units x £6.00 = £300.00  
Total Energy Charges: £1,200.00

Would you like to calculate energy charges for custom values?  
Enter 'y' for interactive calculator, or any other key to exit: y

INTERACTIVE ENERGY CHARGES CALCULATOR

Enter units consumed: 120  
Customer Types:  
1. Domestic  
2. Commercial  
3. Industrial  
Select customer type (1/2/3): 1

CALCULATION RESULT

Units Consumed: 120.00 units  
Customer Type: Domestic  
Energy Charges: £440.00

Thank you for using Energy Charges Calculator!

(lvmv) [base] boda.manishwar@BodaManishwar-Laptop AIC %

## **CODE EXPLANATION:**

This script demonstrates the calculation of Energy Charges (EC) based on units consumed and customer type. It compares two AI-generated implementations—an original version and an improved, optimized version—to ensure both produce the same results. The program uses conditional logic for Domestic, Commercial, and Industrial consumers, displays comparison results in a tabular format, and allows interactive user input for real-time calculation. This task highlights the use of AI tools to optimize conditional statements while maintaining correct billing logic.

### **Q) 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

- Includes meaningful output

- Function-based Python program
  - Correct EC and FC values
  - Screenshots of AI-assisted function generation
  - Test cases with outputs

## PROMPT:

Create a Python billing program using functions to calculate and return Energy Charges and Fixed Charges with proper comments and sample outputs.

CODE:

AI

Modular billing design with AI

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

Scenario

4 of 4 To-dos Completed

**SUMMARY**

- ✓ All functions are modular and reusable
- ✓ calculate\_energy\_charges() returns correct EC values
- ✓ calculate\_fixed\_charges() returns correct FC values
- ✓ Functions can be used independently or combined
- ✓ All test cases executed successfully!

Creating a separate test file with additional test cases and updating the README:

Task3\_Test\_Cases.py +181

Task 3: Test Cases for Modular Bill

This file contains comprehensive test cases for the modular bill design.

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodila.manishwar/Downloads/AI Assistant coding/AIC" && python Task3_Test_Cases.py
```

Plan, @ for context, / for commands

Auto → Review

Task3\_Billing\_Modular.py

```
def calculate_fixed_charges(consumer_type, connection_load_kw):
    # Default to Residential if invalid type is provided
    rate = fixed_charge_rates.get(consumer_type, fixed_charge_rates['Residential'])

    # Calculate Fixed Charges: FC = Rate per kW × Connection Load
    fixed_charges = rate * connection_load_kw

    return fixed_charges
```

def calculate\_total\_bill(units\_consumed, rate\_per\_unit, consumer\_type, connection\_load\_kw):
 """
 Calculates the total bill by combining Energy Charges and Fixed Charges.

 This function demonstrates how the modular functions can be combined to calculate a complete bill for a consumer.
 """

 Args:
 units\_consumed (float): The number of units (kWh) consumed
 rate\_per\_unit (float): The rate charged per unit
 consumer\_type (str): Type of consumer ('Residential', 'Commercial', 'Industrial')
 connection\_load\_kw (float): The connection load in kilowatts

 Returns:
 dict: A dictionary containing EC, FC, and Total Bill amount

 Example:
 >>> result = calculate\_total\_bill(100, 5.5, 'Residential', 5.0)
 >>> result['Total']
 886.0
 """

 # Use the modular functions to calculate individual charges
 ec = calculate\_energy\_charges(units\_consumed, rate\_per\_unit)
 fc = calculate\_fixed\_charges(consumer\_type, connection\_load\_kw)

 # Calculate total bill
 total\_bill = ec + fc

 # Return all values in a structured format
 return {
 'Energy\_Charges': ec,
 'Fixed\_Charges': fc,
 'Total': total\_bill
 }

def display\_bill\_details(consumer\_name, units\_consumed, rate\_per\_unit, consumer\_type, connection\_load\_kw):
 """
 Displays a formatted bill with all charge details.

 This function demonstrates the reusability of the billing functions by creating a formatted output for any consumer.
 """

 Args:
 consumer\_name (str): Name of the consumer
 units\_consumed (float): The number of units (kWh) consumed
 rate\_per\_unit (float): The rate charged per unit
 consumer\_type (str): Type of consumer ('Residential', 'Commercial', 'Industrial')
 connection\_load\_kw (float): The connection load in kilowatts

 Returns:
 str: A formatted bill string

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodila.manishwar/Downloads/AI Assistant coding/AIC" && python Task3_Test_Cases.py
```

Plan, @ for context, / for commands

Auto → Review

Task3\_Billing\_Modular.py

```
def calculate_energy_charges(units_consumed, rate_per_unit):
    """
    Calculates the Energy Charges (EC) based on units consumed and rate per unit.

    Energy Charges are calculated by multiplying the units consumed by the rate per unit.
    This function is designed to be reusable for multiple consumers with different consumption patterns and rates.
    """

    Args:
        units_consumed (float): The number of units (kWh) consumed by the consumer
        rate_per_unit (float): The rate charged per unit (in currency per kWh)

    Returns:
        float: The total Energy Charges (EC) calculated as units_consumed * rate_per_unit

    Example:
        >>> calculate_energy_charges(100, 5.5)
        550.0
    """

    # Calculate Energy Charges: EC = Units Consumed × Rate per Unit
    energy_charges = units_consumed * rate_per_unit

    return energy_charges
```

def calculate\_fixed\_charges(consumer\_type, connection\_load\_kw):
 """
 Calculates the Fixed Charges (FC) based on consumer type and connection load.

 Fixed Charges are a standard amount charged regardless of consumption.
 Different consumer types (Residential, Commercial, Industrial) have different fixed charge rates. The charge is also proportional to the connection load.
 """

 Args:
 consumer\_type (str): Type of consumer - 'Residential', 'Commercial', or 'Industrial'
 connection\_load\_kw (float): The connection load in kilowatts (kW)

 Returns:
 float: The total Fixed Charges (FC) based on consumer type and load

Example:

```
>>> calculate_fixed_charges('Residential', 5.0)
250.0
"""

# Define fixed charge rates per kW for different consumer types
# These rates are in currency per kW per month
fixed_charge_rates = {
    'Residential': 50.0,      # £50 per kW for residential consumers
    'Commercial': 100.0,     # £100 per kW for commercial consumers
    'Industrial': 150.0       # £150 per kW for industrial consumers
}
```

# Get the rate for the specified consumer type

Cursor Tab Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

AI

Modular billing design with AI

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

Scenario

4 of 4 To-dos Completed

**SUMMARY**

- ✓ All functions are modular and reusable
- ✓ calculate\_energy\_charges() returns correct EC values
- ✓ calculate\_fixed\_charges() returns correct FC values
- ✓ Functions can be used independently or combined
- ✓ All test cases executed successfully!

Creating a separate test file with additional test cases and updating the README:

Task3\_Test\_Cases.py +181

Task 3: Test Cases for Modular Bill

This file contains comprehensive test cases for the modular bill design.

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodila.manishwar/Downloads/AI Assistant coding/AIC" && python Task3_Test_Cases.py
```

Plan, @ for context, / for commands

Auto → Review

Task3\_Billing\_Modular.py

```
def calculate_fixed_charges(consumer_type, connection_load_kw):
    # Default to Residential if invalid type is provided
    rate = fixed_charge_rates.get(consumer_type, fixed_charge_rates['Residential'])

    # Calculate Fixed Charges: FC = Rate per kW × Connection Load
    fixed_charges = rate * connection_load_kw

    return fixed_charges
```

def calculate\_total\_bill(units\_consumed, rate\_per\_unit, consumer\_type, connection\_load\_kw):
 """
 Calculates the total bill by combining Energy Charges and Fixed Charges.

 This function demonstrates how the modular functions can be combined to calculate a complete bill for a consumer.
 """

 Args:
 units\_consumed (float): The number of units (kWh) consumed
 rate\_per\_unit (float): The rate charged per unit
 consumer\_type (str): Type of consumer ('Residential', 'Commercial', 'Industrial')
 connection\_load\_kw (float): The connection load in kilowatts

 Returns:
 dict: A dictionary containing EC, FC, and Total Bill amount

 Example:
 >>> result = calculate\_total\_bill(100, 5.5, 'Residential', 5.0)
 >>> result['Total']
 886.0
 """

 # Use the modular functions to calculate individual charges
 ec = calculate\_energy\_charges(units\_consumed, rate\_per\_unit)
 fc = calculate\_fixed\_charges(consumer\_type, connection\_load\_kw)

 # Calculate total bill
 total\_bill = ec + fc

 # Return all values in a structured format
 return {
 'Energy\_Charges': ec,
 'Fixed\_Charges': fc,
 'Total': total\_bill
 }

def display\_bill\_details(consumer\_name, units\_consumed, rate\_per\_unit, consumer\_type, connection\_load\_kw):
 """
 Displays a formatted bill with all charge details.

 This function demonstrates the reusability of the billing functions by creating a formatted output for any consumer.
 """

 Args:
 consumer\_name (str): Name of the consumer
 units\_consumed (float): The number of units (kWh) consumed
 rate\_per\_unit (float): The rate charged per unit
 consumer\_type (str): Type of consumer ('Residential', 'Commercial', 'Industrial')
 connection\_load\_kw (float): The connection load in kilowatts

 Returns:
 str: A formatted bill string

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodila.manishwar/Downloads/AI Assistant coding/AIC" && python Task3_Test_Cases.py
```

Plan, @ for context, / for commands

Auto → Review

Task3\_Billing\_Modular.py

```
def calculate_energy_charges(units_consumed, rate_per_unit):
    """
    Calculates the Energy Charges (EC) based on units consumed and rate per unit.

    Energy Charges are calculated by multiplying the units consumed by the rate per unit.
    This function is designed to be reusable for multiple consumers with different consumption patterns and rates.
    """

    Args:
        units_consumed (float): The number of units (kWh) consumed by the consumer
        rate_per_unit (float): The rate charged per unit (in currency per kWh)

    Returns:
        float: The total Energy Charges (EC) calculated as units_consumed * rate_per_unit

    Example:
        >>> calculate_energy_charges(100, 5.5)
        550.0
    """

    # Calculate Energy Charges: EC = Units Consumed × Rate per Unit
    energy_charges = units_consumed * rate_per_unit

    return energy_charges
```

def calculate\_fixed\_charges(consumer\_type, connection\_load\_kw):
 """
 Calculates the Fixed Charges (FC) based on consumer type and connection load.

 Fixed Charges are a standard amount charged regardless of consumption.
 Different consumer types (Residential, Commercial, Industrial) have different fixed charge rates. The charge is also proportional to the connection load.
 """

 Args:
 consumer\_type (str): Type of consumer - 'Residential', 'Commercial', or 'Industrial'
 connection\_load\_kw (float): The connection load in kilowatts (kW)

 Returns:
 float: The total Fixed Charges (FC) based on consumer type and load

Example:

```
>>> calculate_fixed_charges('Residential', 5.0)
250.0
"""

# Define fixed charge rates per kW for different consumer types
# These rates are in currency per kW per month
fixed_charge_rates = {
    'Residential': 50.0,      # £50 per kW for residential consumers
    'Commercial': 100.0,     # £100 per kW for commercial consumers
    'Industrial': 150.0       # £150 per kW for industrial consumers
}
```

# Get the rate for the specified consumer type

Cursor Tab Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

Modular billing design with AI

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

Scenario

4 of 4 To-dos Completed

**SUMMARY**

- >All functions are modular and reusable
- `calculate_energy_charges()` returns correct EC values
- `calculate_fixed_charges()` returns correct FC values
- Functions can be used independently or combined
- All test cases executed successfully!

Creating a separate test file with additional test cases and updating the README:

```
Task3_Test_Cases.py +181
"""
Task 3: Test Cases for Modular Bill
This file contains comprehensive test cases for the modular billing system.

Auto-Ran command in sandbox: cd python
$ cd "/Users/bodla.manishwar/Downloads/AT Assistant coding/AIC" && python Task3_Test_Cases.py
> O File Review
Plan, @ for context, / for commands
@@ Auto @@ + @@ ?
```

Task3\_Billing\_Modular.py

```
def main():
    ec1 = calculate_energy_charges(units1, rate1)
    fc1 = calculate_fixed_charges(type1, load1)

    print("Consumer: (consumer1_name)")
    print("Units Consumed: (units1) kWh")
    print("Rate per Unit: (rate1)")
    print("Consumer Type: (type1)")
    print("Connection Load: (load1) kW")
    print("Energy Charges (EC): (ec1:.2f)")
    print("Fixed Charges (FC): (fc1:.2f)")
    print("Total Bill: (ec1 + fc1:.2f)")

    # Test Case 2: Commercial Consumer
    print("\n" + "=" * 70)
    print("TEST CASE 2: Commercial Consumer")
    print("=" * 70)
    consumer2_name = "ABC Store"
    units2 = 500.0
    rate2 = 7.0
    type2 = "Commercial"
    load2 = 10.0

    ec2 = calculate_energy_charges(units2, rate2)
    fc2 = calculate_fixed_charges(type2, load2)

    print("Consumer: (consumer2_name)")
    print("Units Consumed: (units2) kWh")
    print("Rate per Unit: (rate2)")
    print("Consumer Type: (type2)")
    print("Connection Load: (load2) kW")
    print("Energy Charges (EC): (ec2:.2f)")
    print("Fixed Charges (FC): (fc2:.2f)")
    print("Total Bill: (ec2 + fc2:.2f)")

    # Test Case 3: Industrial Consumer
    print("\n" + "=" * 70)
    print("TEST CASE 3: Industrial Consumer")
    print("=" * 70)
    consumer3_name = "XYZ Manufacturing"
    units3 = 2000.0
    rate3 = 8.5
    type3 = "Industrial"
    load3 = 50.0

    ec3 = calculate_energy_charges(units3, rate3)
    fc3 = calculate_fixed_charges(type3, load3)

    print("Consumer: (consumer3_name)")
    print("Units Consumed: (units3) kWh")
    print("Rate per Unit: (rate3)")
    print("Consumer Type: (type3)")

    # Calculate all charges using the modular functions
    bill_details = calculate_total_bill(units_consumed, rate_per_unit,
                                         consumer_type, connection_load_kw)

    # Display formatted bill
    print("\n" + "=" * 70)
    print("ELECTRICITY BILL - (consumer_name.upper())")
    print("=" * 70)
    print("Consumer Type: (consumer_type)")
    print("Connection Load: (connection_load_kw) kW")
    print("Units Consumed: (units_consumed) kWh")
    print("Rate per Unit: (rate_per_unit:.2f)")
    print("=" * 70)
    print("Energy Charges (EC): (bill_details['Energy_Charges']:.2f)")
    print("Fixed Charges (FC): (bill_details['Fixed_Charges']:.2f)")
    print("=" * 70)
    print("TOTAL BILL: (bill_details['Total']:.2f)")
    print("\n" + "=" * 70 + "\n")

def display_bill_details(consumer_name, units_consumed, rate_per_unit,
                        consumer_type, connection_load_kw):
    """ Main function to demonstrate the modular billing system with test cases.

    This function runs multiple test cases to show how the reusable functions
    work for different consumers with varying consumption patterns.
    """
    print("\n" + "=" * 70)
    print("TASK 3: MODULAR BILLING DESIGN USING FUNCTIONS")
    print("\n" + "=" * 70)
    print("This program demonstrates reusable functions for billing calculations:")
    print("* calculate_energy_charges() - Calculates EC based on units and rates")
    print("* calculate_fixed_charges() - Calculates FC based on consumer type and load")
    print("\n" + "=" * 70 + "\n")

    # Test Case 1: Residential Consumer
    print("\n" + "=" * 70)
    print("TEST CASE 1: Residential Consumer")
    print("\n" + "=" * 70)
    consumer1_name = "John Doe"
    units1 = 150.0
    rate1 = 5.5
    type1 = "Residential"
    load1 = 3.0
```

Modular billing design with AI

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

Scenario

4 of 4 To-dos Completed

**SUMMARY**

- All functions are modular and reusable
- `calculate_energy_charges()` returns correct EC values
- `calculate_fixed_charges()` returns correct FC values
- Functions can be used independently or combined
- All test cases executed successfully!

Creating a separate test file with additional test cases and updating the README:

```
Task3_Test_Cases.py +181
"""
Task 3: Test Cases for Modular Bill
This file contains comprehensive test cases for the modular billing system.

Auto-Ran command in sandbox: cd python
$ cd "/Users/bodla.manishwar/Downloads/AT Assistant coding/AIC" && python Task3_Test_Cases.py
> O File Review
Plan, @ for context, / for commands
@@ Auto @@ + @@ ?
```

Task3\_Billing\_Modular.py

```
def Task3_Billing_Modular.py > ...
def display_bill_details(consumer_name, units_consumed, rate_per_unit,
                        consumer_type, connection_load_kw):
    """ Main function to demonstrate the modular billing system with test cases.

    This function runs multiple test cases to show how the reusable functions
    work for different consumers with varying consumption patterns.
    """
    print("\n" + "=" * 70)
    print("TASK 3: MODULAR BILLING DESIGN USING FUNCTIONS")
    print("\n" + "=" * 70)
    print("This program demonstrates reusable functions for billing calculations:")
    print("* calculate_energy_charges() - Calculates EC based on units and rates")
    print("* calculate_fixed_charges() - Calculates FC based on consumer type and load")
    print("\n" + "=" * 70 + "\n")

    # Test Case 1: Residential Consumer
    print("\n" + "=" * 70)
    print("TEST CASE 1: Residential Consumer")
    print("\n" + "=" * 70)
    consumer1_name = "John Doe"
    units1 = 150.0
    rate1 = 5.5
    type1 = "Residential"
    load1 = 3.0
```

The screenshot shows the AI Assistant interface with the following details:

- Code Review:** A code editor window titled "Task3\_Billing\_Modular.py" containing Python code for calculating energy bills.
- Test Cases:** A separate window titled "Task3\_Test\_Cases.py" showing 181 test cases.
- README:** A section with a summary of the task's goals and requirements.
- Auto-Ran command:** An auto-generated command to run the tests in a terminal.
- Plan:** A sidebar with a plan for context and commands.
- Outline and Timeline:** Navigation tools on the right side of the interface.

## OUTPUT:

The terminal output displays the results of the test cases and bill calculations:

```

TEST CASE 2: Commercial Consumer
Consumer: ABC Store
Units Consumed: 500.0 kWh
Rate per Unit: ₹7.0
Consumer Type: Commercial
Connection Load: 50.0 kW
Energy Charges (EC): ₹3500.00
Fixed Charges (FC): ₹1600.00
Total Bill: ₹4500.00

TEST CASE 3: Industrial Consumer
Consumer: XYZ Manufacturing
Units Consumed: 2000.0 kWh
Rate per Unit: ₹8.0
Consumer Type: Industrial
Connection Load: 50.0 kW
Energy Charges (EC): ₹17000.00
Fixed Charges (FC): ₹7500.00
Total Bill: ₹24500.00

TEST CASE 4: Using calculate_total_bill() Function
Using calculate_total_bill(250.0, 6.0, 'Residential', 4.0):
Energy Charges (EC): ₹1500.00
Fixed Charges (FC): ₹7200.00
Total Bill: ₹17700.00

TEST CASE 5: Formatted Bill Display
ELECTRICITY BILL - SARAH SMITH
Consumer Type: Residential
Connection Load: 2.5 kW
Units Consumed: 180.0 kWh
Rate per Unit: ₹5.50
Energy Charges (EC): ₹990.00
Fixed Charges (FC): ₹125.00
TOTAL BILL: ₹1115.00

SUMMARY
All functions are modular and reusable
calculate_energy_charges() returns correct EC values
calculate_fixed_charges() returns correct FC values
Functions can be used independently or combined
All test cases executed successfully!

```

```

source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
# [base] bodla.manishwar@BodaManishwar-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
# [base] bodla.manishwar@BodaManishwar-Laptop AIC % cd "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC"
# [base] bodla.manishwar@BodaManishwar-Laptop AIC % python Task3_Billing_AIC.py
=====
TASK 3: MODULAR BILLING DESIGN USING FUNCTIONS
=====

This program demonstrates reusable functions for billing calculations:
+ calculate_energy_charges() - Calculates EC based on units and rate
+ calculate_fixed_charges() - Calculates FC based on consumer type and load
=====

TEST CASE 1: Residential Consumer
=====

Consumer: John Doe
Units Consumed: 150.0 kWh
Rate per Unit: ₹7.0
Consumer Type: Residential
Connection Load: 3.0 kW

Energy Charges (EC): ₹1050.00
Fixed Charges (FC): ₹150.00
Total Bill: ₹1200.00
=====

TEST CASE 2: Commercial Consumer
=====

Consumer: ABC Store
Units Consumed: 2000.0 kWh
Rate per Unit: ₹7.0
Consumer Type: Commercial
Connection Load: 10.0 kW

Energy Charges (EC): ₹14000.00
Fixed Charges (FC): ₹1000.00
Total Bill: ₹15000.00
=====

TEST CASE 3: Industrial Consumer
=====

Consumer: XYZ Manufacturing
Units Consumed: 20000.0 kWh
Rate per Unit: ₹8.5
Consumer Type: Industrial
Connection Load: 50.0 kW

Energy Charge (EC): ₹175000.00
Fixed Charges (FC): ₹15000.00
Total Bill: ₹190000.00
=====

TEST CASE 4: Using calculate_total_bill() Function
=====

Using calculate_total_bill(250.0, 6.0, 'Residential', 4.0):
=====
Energy Charges (EC): ₹1500.00
Fixed Charges (FC): ₹200.00
Total Bill: ₹1700.00
=====

TEST CASE 5: Formatted Bill Display
=====

Consumer: ABC Store
Units Consumed: 2000.0 kWh
Rate per Unit: ₹7.0
Consumer Type: Commercial
Connection Load: 10.0 kW

Energy Charges (EC): ₹14000.00
Fixed Charges (FC): ₹1000.00
Total Bill: ₹15000.00
=====

XK to generate command

```

## CODE EXPLANATION:

This program demonstrates a modular electricity billing system using user-defined functions. Separate reusable functions are created to calculate Energy Charges (EC) based on units consumed and rate per unit, and Fixed Charges (FC) based on consumer type and connection load. These functions return calculated values, allowing them to be reused for multiple consumers. The program includes multiple test cases to verify correctness and shows how modular design improves readability, reusability, and maintainability of billing logic using AI assistance.

### Q) 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”

#### Expected Output

- Individual charge values printed
- Correct duty calculation
- Well-structured output
- Verified intermediate results

### PROMPT:

Modify the Python billing program to calculate FC, CC, and Electricity Duty (percentage of EC) and display all charges clearly.

### CODE:

**Electricity bill additional charges**

1m - +214 - Auto

ask 4: Calculation of Additional Charges Scenario

Searching for existing electricity billing code:

Explored 3 searches

No existing electricity billing code found. Creating a program to calculate additional charges (FC, CC, ED) for electricity bills:

Task 4: Calculation of Additional C This program calculates electricity charges.

Testing the program to verify it works correctly:

Auto-Ran command in sandbox: cd python \$ cd "/Users/bodla.manishwar/Downloads/AT Assistant coding/AIC" && python Task4\_Electricity\_Billing.py

Intermediate results displayed for verification ✓ Success

No linter errors found in Task4\_Electricity\_B... Program created and tested. Summary:

**Task 4: Calculation of Additional Charges - Complete**

The program calculates electricity bill charges:

Review Plan, @ for context, / for commands Auto ✓

```

Task4_Electricity_Billing.py ×
def calculate_electricity_duty(energy_charges, duty_percentage=5.0):
    Calculate Electricity Duty (ED) as a percentage of Energy Charges.

    Args:
        energy_charges (float): Energy Charges amount
        duty_percentage (float): Duty percentage (default 5%)
    
    Returns:
        float: Electricity Duty amount
    ed = (energy_charges * duty_percentage) / 100
    return ed

def calculate_total_bill(units_consumed, rate_per_unit, connection_type='domestic', duty_percentage=5.0):
    Calculate complete electricity bill with all charges.

    Args:
        units_consumed (float): Number of electricity units consumed
        rate_per_unit (float): Rate per unit in currency
        connection_type (str): type of connection (domestic, commercial, industrial)
        duty_percentage (float): Electricity duty percentage (default 5%)
    
    Returns:
        dict: Dictionary containing all charge breakdown and total
    # Calculate individual charges
    ec = calculate_energy_charges(units_consumed, rate_per_unit)
    fc = calculate_fixed_charges(connection_type)
    cc = calculate_customer_charges(connection_type)
    ed = calculate_electricity_duty(ec, duty_percentage)

    # Calculate total bill
    total_bill = ec + fc + cc + ed

    return {
        'units_consumed': units_consumed,
        'rate_per_unit': rate_per_unit,
        'connection_type': connection_type,
        'energy_charges': ec,
        'fixed_charges': fc,
        'customer_charges': cc,
        'electricity_duty': ed,
        'duty_percentage': duty_percentage,
        'total_bill': total_bill
    }

def print_bill_breakdown(bill_details):
    Print a well-structured electricity bill breakdown.

```

Cursor Tab: Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

**Electricity bill additional charges**

2m - +214 - Auto

ask 4: Calculation of Additional Charges Scenario

Searching for existing electricity billing code:

Explored 3 searches

No existing electricity billing code found. Creating a program to calculate additional charges (FC, CC, ED) for electricity bills:

Task 4: Calculation of Additional C This program calculates electricity charges.

Testing the program to verify it works correctly:

Auto-Ran command in sandbox: cd python \$ cd "/Users/bodla.manishwar/Downloads/AT Assistant coding/AIC" && python Task4\_Electricity\_Billing.py

Intermediate results displayed for verification ✓ Success

No linter errors found in Task4\_Electricity\_B... Program created and tested. Summary:

**Task 4: Calculation of Additional Charges - Complete**

The program calculates electricity bill charges:

Review Plan, @ for context, / for commands Auto ✓

```

Task4_Electricity_Billing.py ×
def print_bill_breakdown(bill_details):
    Args:
        bill_details (dict): Dictionary containing bill details
    ...
    print("\n" * 70)
    print("ELECTRICITY BILL BREAKDOWN")
    print("\n" * 70)
    print(f"\nConnection Type: {bill_details['connection_type'].upper()}")
    print(f"\nUnits Consumed: {(bill_details['units_consumed']):.2f} units")
    print(f"\nRate per Unit: {(bill_details['rate_per_unit']):.2f}")
    print(f"\nElectricity Duty Rate: {(bill_details['duty_percentage']):.2f}%")
    print("\n" * 70)
    print(f"\nCharge Breakdown:")
    print("\n" * 70)
    print(f"\n{('Charge Type':>25} {'Amount (>15)':>15})")
    print("\n" * 70)
    print(f"\n{('Energy Charges (EC)':>25} {(bill_details['energy_charges']):>15.2f})")
    print(f"\n{('Fixed Charges (FC)':>25} {(bill_details['fixed_charges']):>15.2f})")
    print(f"\n{('Customer Charges (CC)':>25} {(bill_details['customer_charges']):>15.2f})")
    print(f"\n{('Electricity Duty (ED)':>25} {(bill_details['electricity_duty']):>15.2f})")
    print("\n" * 70)
    print(f"\nTOTAL BILL:<25> {(bill_details['total_bill']):>15.2f}")
    print("\n" * 70)
    print("\n" * 70)

    # Print intermediate results for verification
    print("\n" * 70)
    print("INTERMEDIATE CALCULATIONS (For Verification):")
    print("\n" * 70)
    print(f"\nEC = Units [ Rate = {(bill_details['units_consumed']):.2f} ] * {(bill_details['rate_per_unit']):.2f} = {(bill_details['energy_charges']):.2f} (based on {bill_details['connection_type']} connection)")
    print(f"\nFC = {(bill_details['fixed_charges']):.2f} (based on {bill_details['connection_type']} connection)")
    print(f"\nCC = {(bill_details['customer_charges']):.2f} (based on {bill_details['connection_type']} connection)")
    print(f"\nED = EC [ (duty_percentage):.2f ] * {(bill_details['energy_charges']):.2f} = {(bill_details['electricity_duty']):.2f} (based on {bill_details['connection_type']} connection)")
    print(f"\nTotal Bill = FC + CC + ED = {(bill_details['energy_charges']):.2f} + {(bill_details['fixed_charges']):.2f} + {(bill_details['customer_charges']):.2f} + {(bill_details['electricity_duty']):.2f} = {(bill_details['total_bill']):.2f}")
    print("\n" * 70 + "\n")

def main():
    """
    Main function to demonstrate electricity bill calculation.
    """
    print("\n" * 70)
    print("Task 4: Calculation of Additional Charges for Electricity Bills")
    print("\n" * 70)

    # Example 1: Domestic connection
    print("\n" * 70)
    print("Example 1: Domestic Connection")
    print("\n" * 70)
    bill1 = calculate_total_bill(
        units_consumed=150.0,
        rate_per_unit=6.50,
        connection_type='domestic',
        duty_percentage=5.0
    )

```

Cursor Tab: Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

**Electricity bill additional charges**

2m +214 Auto

ask 4: Calculation of Additional Charges Scenario

Searching for existing electricity billing code:

Explored 3 searches

No existing electricity billing code found.

Creating a program to calculate additional charges (FC, CC, ED) for electricity bills:

Task 4: Calculation of Additional C

This program calculates electricity bill charges:

Testing the program to verify it works correctly:

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC" && python Task4_Electricity_Billing.py
```

d  Intermediate results displayed for verification ✓ Success

No linter errors found in Task4\_Electricity\_B...

Program created and tested. Summary:

**Task 4: Calculation of Additional Charges - Complete**

The program calculates electricity bill charges:

> 0 File Review

Plan, @ for context, / for commands

Auto ~ Auto

**Task4\_Electricity\_Billing.py**

```
155 def main():
156     """
157     print_bill_breakdown(bill1)
158
159     # Example 2: Commercial connection
160     print("\nExample 2: Commercial Connection")
161     print("—" * 70)
162     bill2 = calculate_total_bill(
163         units_consumed=500.0,
164         rate_per_unit=5.0,
165         connection_type='commercial',
166         duty_percentage=7.5
167     )
168     print_bill_breakdown(bill2)
169
170     # Example 3: Industrial connection
171     print("\nExample 3: Industrial Connection")
172     print("—" * 70)
173     bill3 = calculate_total_bill(
174         units_consumed=2000.0,
175         rate_per_unit=10.0,
176         connection_type='industrial',
177         duty_percentage=10.0
178     )
179     print_bill_breakdown(bill3)
180
181     # Individual charge values demonstration
182     print("\n" + "—" * 70)
183     print("INDIVIDUAL CHARGE VALUES (Example 1 Breakdown)")
184     print("—" * 70)
185     print(f"FC (Fixed Charges): {bill1['fixed_charges']:.2f}")
186     print(f"CC (Customer Charges): {bill1['customer_charges']:.2f}")
187     print(f"ED (Electricity Duty): {bill1['electricity_duty']:.2f} ({bill1['duty_percentage']:.2f}% of EC)")
188     print("—" * 70)
189
190     print("\n" + "—" * 70)
191     print("All calculations completed successfully!")
192     print("✓ Individual charge values printed")
193     print("✓ Correct duty calculation verified")
194     print("✓ Well-structured output generated")
195     print("✓ Intermediate results displayed for verification")
196     print("—" * 70)
197
198 if __name__ == "__main__":
199     main()
```

AIC Cursor Tab Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

**Electricity bill additional charges**

Now +214 Auto

ask 4: Calculation of Additional Charges Scenario

Searching for existing electricity billing code:

Explored 3 searches

No existing electricity billing code found.

Creating a program to calculate additional charges (FC, CC, ED) for electricity bills:

Task 4: Calculation of Additional C

This program calculates electricity bill charges:

Testing the program to verify it works correctly:

Auto-Ran command in sandbox: cd python

```
$ cd "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC" && python Task4_Electricity_Billing.py
```

d  Intermediate results displayed for verification ✓ Success

No linter errors found in Task4\_Electricity\_B...

Program created and tested. Summary:

**Task 4: Calculation of Additional Charges - Complete**

The program calculates electricity bill charges:

> 0 File Review

Plan, @ for context, / for commands

Auto ~ Auto

**Task4\_Electricity\_Billing.py**

```
14 def calculate_energy_charges(units_consumed, rate_per_unit):
15     """
16     Calculate Energy Charges (EC) based on units consumed.
17
18     Args:
19         units_consumed (float): Number of electricity units consumed
20         rate_per_unit (float): Rate per unit in currency
21
22     Returns:
23         float: Energy Charges amount
24
25     ec = units_consumed * rate_per_unit
26
27     return ec
28
29 def calculate_fixed_charges(connection_type='domestic'):
30     """
31     Calculate Fixed Charges (FC) based on connection type.
32
33     Args:
34         connection_type (str): Type of connection (domestic, commercial, industrial)
35
36     Returns:
37         float: Fixed Charges amount
38
39     fixed_charges_map = {
40         'domestic': 50.0,
41         'commercial': 100.0,
42         'industrial': 200.0
43     }
44
45     return fixed_charges_map.get(connection_type.lower()), 50.0
46
47 def calculate_customer_charges(connection_type='domestic'):
48     """
49     Calculate Customer Charges (CC) based on connection type.
50
51     Args:
52         connection_type (str): Type of connection (domestic, commercial, industrial)
53
54     Returns:
55         float: Customer Charges amount
56
57     customer_charges_map = {
58         'domestic': 25.0,
59         'commercial': 50.0,
60         'industrial': 75.0
61     }
62
63     return customer_charges_map.get(connection_type.lower()), 25.0
64
65 def calculate_electricity_duty(energy_charges, duty_percentage=5.0):
66     """
67
```

AIC Cursor Tab Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv) □

## OUTPUT:

The screenshot shows a Jupyter Notebook interface with several cells and a sidebar.

**Sidebar:**

- Search Agents...
- New Agent
- Agents
- Electricity bill addition... 2m

**File Explorer:**

- AIC
- README\_Task2.md
- README\_Task3.md
- sample\_data.csv
- sample\_output.txt
- task2\_file\_Operatio...
- Task3\_Csv\_Data\_A...
- Task4\_Electricity\_B...

**Code Cells (Task4\_Electricity\_Bill.ipynb):**

- Cell 1:** Electricity bill additional charges  
2m +214 - Auto  
ask 4: Calculation of Additional Charges Scenario  
Electricity Bill Addition
- Cell 2:** Searching for existing electricity billing code:  
Explored 3 searches  
No existing electricity billing code found.  
Creating a program to calculate additional charges (FC, CC, ED) for electricity bills:  
Task4\_Electricity\_Billing.py +214
- Cell 3:** Task 4: Calculation of Additional Charges  
This program calculates electricity bills  
Testing the program to verify it works correctly:  
Auto-Ran command in sandbox: cd python  
\$ cd "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC" && python Task4\_Electricity\_Billing.py  
d  
✓ Intermediate results displayed for verification  
✓ Success  
No linter errors found in Task4\_Electricity\_Bill.ipynb
- Cell 4:** Program created and tested. Summary:  
**Task 4: Calculation of Additional Charges - Complete**  
The program calculates electricity bill charges:  
> 0 File Review  
Plan, @ for context, / for commands  
○ Auto ▾
- Cell 5:** source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"  
"/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task4\_Electricity\_Billing.py"  
(base) bodla.manishwar@BODLA-MANISHWAR-LAPTOP-AIC ~ % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"  
(base) bodla.manishwar@BODLA-MANISHWAR-LAPTOP-AIC ~ % python "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task4\_Electricity\_Billing.py"  
Task 4: Calculation of Additional Charges for Electricity Bills
- Cell 6:** Example 1: Domestic Connection  
**ELECTRICITY BILL BREAKDOWN**  
Connection Type: DOMESTIC  
Units Consumed: 150.00 units  
Rate per Unit: ₹6.50  
Electricity Duty Rate: 5.00%  
**CHARGE BREAKDOWN:**  

Charge Type	Amount (₹)
Energy Charges (EC)	975.00
Fixed Charges (FC)	50.00
Customer Charges (CC)	25.00
Electricity Duty (ED)	48.75
<b>TOTAL BILL</b>	<b>1098.75</b>

  
**INTERMEDIATE CALCULATIONS (For Verification):**  
EC = Units x Rate = 150.00 x ₹6.50 = ₹975.00  
FC = ₹50.00 (based on domestic connection)  
CC = ₹25.00 (based on domestic connection)  
ED = EC x 5.00% = ₹975.00 x 5.00% = ₹48.75  
Total = EC + FC + CC + ED = ₹975.00 + ₹50.00 + ₹25.00 + ₹48.75 = ₹1098.75
- Cell 7:** Example 2: Commercial Connection  
**ELECTRICITY BILL BREAKDOWN**  
Connection Type: COMMERCIAL  
Units Consumed: 400.00 units  
Rate per Unit: ₹8.00  
Electricity Duty Rate: 7.50%  
**CHARGE BREAKDOWN:**  

Charge Type	Amount (₹)
Energy Charges (EC)	4000.00
Fixed Charges (FC)	100.00
Customer Charges (CC)	50.00
Electricity Duty (ED)	300.00

**CODE EXPLANATION:** This program extends the electricity billing system to calculate additional charges such as Fixed Charges (FC), Customer Charges (CC), and Electricity Duty (ED). Energy Charges are calculated based on units consumed, while FC and CC depend on the connection type. Electricity Duty is computed as a percentage of Energy Charges to ensure accurate billing. The program displays each charge separately with a well-structured bill format and shows intermediate calculations for verification, demonstrating improved billing accuracy using modular functions and AI assistance.

## Q) 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:
  - Energy Charges (EC)
  - Fixed Charges (FC)
  - Customer Charges (CC)
  - Electricity Duty (ED)
  - Total Bill Amount
- Analyze the program based on:
  - Accuracy
  - Readability
  - Real-world applicability

### Expected Output

- Complete electricity bill output
- Neatly formatted display
- Sample input/output
- Short analysis paragraph

### PROMPT:

Create a Python program to generate a final electricity bill showing EC, FC, CC, ED, and total amount, with formatted output and a short analysis.

### CODE:

The screenshot shows a code editor interface with several tabs and panels. The main area displays Python code for a class named ElectricityBillCalculator. The code includes methods for calculating electricity duty, total bill, and currency formatting. A sidebar on the left shows a project structure with files like Task5\_Final\_Bill\_Generation.py, README\_Task5.md, and sample\_data.csv. Another sidebar on the right shows a file tree with various project files and folders. The bottom status bar indicates the current tab, line number, column number, and file path.

```
18     class ElectricityBillCalculator:
19         def calculate_electricity_duty(self, ec: float, fc: float, cc: float) -> float:
20             Args:
21                 ec (float): Energy Charges
22                 fc (float): Fixed Charges
23                 cc (float): Customer Charges
24
25             Returns:
26                 float: Electricity Duty amount
27             """
28             base_amount = ec + fc + cc
29             return base_amount * self.electricity_duty_rate
30
31     def calculate_total_bill(self, units_consumed: float) -> Dict[str, float]:
32         """
33             Calculate all bill components and total bill amount.
34
35             Args:
36                 units_consumed (float): Number of kWh units consumed
37
38             Returns:
39                 dict: Dictionary containing all bill components and total
40             """
41
42         # Calculate individual components
43         ec = self.calculate_energy_charges(units_consumed)
44         fc = self.calculate_fixed_charges()
45         cc = self.calculate_customer_charges()
46         ed = self.calculate_electricity_duty(ec, fc, cc)
47
48         # calculate total bill
49         total_bill = ec + fc + cc + ed
50
51         return {
52             'units_consumed': units_consumed,
53             'energy_charges': ec,
54             'fixed_charges': fc,
55             'customer_charges': cc,
56             'electricity_duty': ed,
57             'total_bill': total_bill
58         }
59
60     def format_currency(self, amount: float) -> str:
61         """
62             Format amount as Indian Rupees currency.
63
64             Args:
65                 amount (float): Amount to format
66
67             Returns:
68                 str: Formatted currency string
69             """
70             return f"₹{amount:.2f}"
```

```

18 class ElectricityBillCalculator:
19     def display_bill(self, bill_data: Dict[str, float], customer_name: str = "Customer",
20                     customer_id: str = "CUST001", billing_month: str = None) -> None:
21         """
22             ...
23             Display the electricity bill in a formatted, professional manner.
24         """
25         Args:
26             bill_data (dict): Dictionary containing bill components
27             customer_name (str): Name of the customer
28             customer_id (str): Customer ID
29             billing_month (str): Billing month (defaults to current month)
30         """
31         if billing_month is None:
32             billing_month = datetime.now().strftime("%B %Y")
33
34         # Extract values
35         units = bill_data["units_consumed"]
36         ec = bill_data["energy_charges"]
37         fc = bill_data["fixed_charges"]
38         cc = bill_data["customer_charges"]
39         ed = bill_data["electricity_duty"]
40         total = bill_data["total_bill"]
41
42         # Print bill header
43         print("\n" * 70)
44         print(" " * 20 + "ELECTRICITY BILL")
45         print("\n" * 70)
46         print(f"Customer Name:{customer_name}")
47         print(f"Customer ID:{customer_id}")
48         print(f"Billing Month:{billing_month}")
49         print(f"Bill Date:{datetime.now().strftime('%d-%m-%Y')}")
50         print("\n" * 70)
51
52         # Print consumption details
53         print("\n" * 70)
54         print("CONSUMPTION DETAILS")
55         print("\n" * 70)
56         print(f"Units Consumed (KWh):{units} (units:.2f)")
57         print(f"Rate per Unit:{ed} ({self.format_currency(self, ed)})")
58         print("\n" * 70)
59
60         # Print bill breakdown
61         print("\n" * 70)
62         print("BILL BREAKDOWN")
63         print("\n" * 70)
64         print(f"1. Energy Charges (EC):{ec} ({self.format_currency(ec)})")
65         print(f"   ({units:.2f} units * {self.format_currency(self.energy_rate_per_unit)}))")
66         print("\n" * 70)
67         print(f"2. Fixed Charged (FC):{fc} ({self.format_currency(fc)})")
68         print("\n" * 70)
69         print(f"3. Customer Charges (CC):{cc} ({self.format_currency(cc)})")
70         print("\n" * 70)
71         print(f"4. Electricity Duty (ED):{ed} ({self.format_currency(ed)})")
72         print(f"   (15% of EC + FC + CC)")
73         print("\n" * 70)

```

```

18 class ElectricityBillCalculator:
19     def display_bill(self, bill_data: Dict[str, float], customer_name: str = "Customer",
20                     customer_id: str = "CUST001", billing_month: str = None) -> None:
21         """
22             ...
23             Display the electricity bill in a formatted, professional manner.
24         """
25         Args:
26             bill_data (dict): Dictionary containing bill components
27             customer_name (str): Name of the customer
28             customer_id (str): Customer ID
29             billing_month (str): Billing month (defaults to current month)
30         """
31         if billing_month is None:
32             billing_month = datetime.now().strftime("%B %Y")
33
34         # Print total
35         print("\n" * 70)
36         print("TOTAL BILL AMOUNT:{total} ({self.format_currency(total)})")
37         print("\n" * 70)
38         print()
39
40         # Print calculation summary
41         print("\n" * 70)
42         print("CALCULATION SUMMARY")
43         print("\n" * 70)
44         print(f"Total Bill = EC + FC + CC + ED")
45         print(f"Total Bill = {self.format_currency(ec)} + {self.format_currency(fc)} + "
46               f"{self.format_currency(cc)} + {self.format_currency(ed)}")
47         print(f"Total Bill = {self.format_currency(total)}")
48         print("\n" * 70)
49
50         def get_user_input() -> Tuple[str, str, float, str]:
51             """
52                 tuple: (customer_name, customer_id, units_consumed, billing_month)
53             """
54             print("\n" * 70)
55             print(" " * 15 + "ELECTRICITY BILL CALCULATOR")
56             print("\n" * 70)
57             print()
58
59             customer_name = input("Enter Customer Name: ").strip() or "John Doe"
60             customer_id = input("Enter Customer ID: ").strip() or "CUST001"
61
62             while True:
63                 try:
64                     units_input = input("Enter Units Consumed (KWh): ").strip()
65                     units_consumed = float(units_input)
66                     if units_consumed < 0:
67                         print("X Units consumed cannot be negative. Please try again.")
68                         continue
69                     break
70                 except ValueError:
71                     print("X Invalid input. Please enter a valid number.")
72
73             billing_month = input("Enter Billing Month (e.g., January 2024): ").strip()
74             if not billing_month:
75                 billing_month = datetime.now().strftime("%B %Y")
76
77             return customer_name, customer_id, units_consumed, billing_month
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217

```

Electricity bill generation and analysis

Task 5: Final Bill Generation and Output Analysis Scenario

3m +555 Auto

Agents Electricity bill general... 3m

3 of 3 To-dos Completed

Reviewing existing files to understand the structure and calculations.

Explored 5 files

Searching for existing electricity bill calculation code to understand the components:

Exploded 4 searches

Creating Task 5: final bill generation and output analysis. Building a Python application that calculates and displays all bill components.

Checked to-do list

Task 5: Final Bill Generation and C

This program calculates and displays

Creating the README and testing the application:

README\_Task5.md (new) +206

# Task 5: Final Bill Generation and C

This project implements a comprehen

Checked to-do list

No linter errors found in Task5\_Final\_Bill\_G...

Testing the application to verify it works

Review Plan, @ for context, / for commands

... Auto

Task5\_Final\_Bill\_Generation.py

```

218 def run_sample_cases():
219     """
220         Run sample test cases to demonstrate the bill calculation.
221     """
222     calculator = ElectricityBillCalculator()
223
224     print("\n" * 70)
225     print(" " * 20 + "SAMPLE BILL CASES")
226     print(" " * 70)
227
228     # Sample Case 1: Low consumption
229     print("\n" * 70)
230     print("SAMPLE CASE 1: Low Consumption (150 units)")
231     print(" " * 70)
232     bill1 = calculator.calculate_total_bill(150)
233     calculator.display_bill(bill1, "Alice Johnson", "CUST001", "January 2024")
234
235     # Sample Case 2: Medium consumption
236     print("\n" * 70)
237     print("SAMPLE CASE 2: Medium Consumption (350 units)")
238     print(" " * 70)
239     bill2 = calculator.calculate_total_bill(350)
240     calculator.display_bill(bill2, "Bob Smith", "CUST002", "January 2024")
241
242     # Sample Case 3: High consumption
243     print("\n" * 70)
244     print("SAMPLE CASE 3: High Consumption (750 units)")
245     print(" " * 70)
246     bill3 = calculator.calculate_total_bill(750)
247     calculator.display_bill(bill3, "Charlie Brown", "CUST003", "January 2024")
248
249
250 def print_analysis():
251     """
252         Print program analysis based on accuracy, readability, and real-world applicability.
253     """
254     print("\n" * 70)
255     print(" " * 20 + "PROGRAM ANALYSIS")
256     print(" " * 70)
257     print()
258
259     analysis = """
260 ACCURACY:
261
262 ✓ The program uses precise floating-point calculations for all bill components
263 ✓ Energy consumption is measured in units of kWh per month
264 ✓ Electricity duty is calculated as a percentage of the base amount (EC + FC + CC)
265 ✓ Total Bill is the sum of all components: EC + FC + CC + ED
266 ✓ All calculations follow standard electricity billing practices
267 ✓ Currency formatting ensures proper display with two decimal places
268
269 READABILITY:
270
271 """
272
273     print(analysis)
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349

```

AIC Review Next File

Review Outline Timeline

Cursor Tab Ln 1, Col 1 Spaces: 4 UTF-8 LF Python 3.9.6 ('venv': venv)

Electricity bill generation and analysis

Task 5: Final Bill Generation and Output Analysis Scenario

3m +555 Auto

Agents Electricity bill general... 3m

3 of 3 To-dos Completed

Reviewing existing files to understand the structure and calculations.

Explored 5 files

Searching for existing electricity bill calculation code to understand the components:

Exploded 4 searches

Creating Task 5: final bill generation and output analysis. Building a Python application that calculates and displays all bill components.

Checked to-do list

Task 5: Final Bill Generation and C

This program calculates and displays

Creating the README and testing the application:

README\_Task5.md (new) +206

# Task 5: Final Bill Generation and C

This project implements a comprehen

Checked to-do list

No linter errors found in Task5\_Final\_Bill\_G...

Testing the application to verify it works

Review Plan, @ for context, / for commands

... Auto

Task5\_Final\_Bill\_Generation.py

```

250 def print_analysis():
251     print(analysis)
252     print(" " * 70)
253     print()
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349

```

AIC Review Next File

Review Outline Timeline

Cursor Tab Ln 257, Col 12 Spaces: 4 UTF-8 LF () Python 3.9.6 ('venv': venv)

```

# Task5_Final_Bill_Generation.py

from datetime import datetime
from typing import Dict, Tuple

class ElectricityBillCalculator:
    """
    A comprehensive electricity bill calculator that computes all bill components.
    """

    def __init__(self):
        """
        Initialize the calculator with default tariff rates.
        # Tariff rates (can be customized)
        self.energy_rate_per_unit = 6.50 # ₹ per kWh
        self.fixed_charge_per_month = 50.00 # ₹ per month
        self.customer_charge = 30.00 # ₹ per month
        self.electricity_duty_rate = 0.15 # 15% of (EC + FC + CC)
        """

    def calculate_energy_charges(self, units_consumed: float) -> float:
        """
        Calculate Energy Charges based on units consumed.
        Args:
            units_consumed (float): Number of kWh units consumed
        Returns:
            float: Energy Charges amount
        """
        return units_consumed * self.energy_rate_per_unit

    def calculate_fixed_charges(self) -> float:
        """
        Calculate Fixed Charges (monthly fixed cost).
        Returns:
            float: Fixed Charges amount
        """
        return self.fixed_charge_per_month

    def calculate_customer_charges(self) -> float:
        """
        Calculate Customer Charges (service charges).
        Returns:
            float: Customer Charges amount
        """
        return self.customer_charge

    def calculate_electricity_duty(self, ec: float, fc: float, cc: float) -> float:
        """
        Calculate Electricity Duty as a percentage of (EC + FC + CC).
        Args:
            ec: float
            fc: float
            cc: float
        """
        return (ec + fc + cc) * self.electricity_duty_rate

```

## OUTPUT:

```

Electricity bill generation and a...
4m +555 Auto
Task 5: Final Bill Generation and
Output Analysis
Scenario
This final application will present...
○ 3 of 3 To-dos Completed

Reviewing existing files to understand the
structure and calculations.
Explored 5 files.
Searching for existing electricity bill
calculation code to understand the
components:
Explored 4 searches.
Creating Task 5: final bill generation and
output analysis. Building a Python
application that calculates and displays all
bill components.
Checked-to-do list
..._5_Final_Bill_Generation.py +349
Task 5: Final Bill Generation and C...
This program calculates and displays...
Creating the README and testing the
application:
README_Task5.md (new) +206
# Task 5: Final Bill Generation and C...
This project implements a comprehen...
Checked-to-do list
No inter errors found in Task5_Final_Bill_G...
Testing the application to verify it works
> 0 File Undo All Keep All Review
Plan, @ for context, / for commands
Auto

```

```

source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
(base) bodla.manishwar@bodlaManishwar-Laptop AIC % source "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/activate"
(base) bodla.manishwar@bodlaManishwar-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/Task5_Final_Bill_Generation.py"
=====
ELECTRICITY BILL GENERATION SYSTEM
=====

Choose an option:
1. Calculate bill with custom input
2. View sample bill cases
3. View program analysis
4. Exit

Enter your choice (1-4): 1
=====
ELECTRICITY BILL CALCULATOR
=====

Enter Customer Name: Manishwar
Enter Customer ID: 1276
Enter Units Consumed (kWh): 120
Enter Billing Month (e.g., January 2024): August 2024
=====
ELECTRICITY BILL
=====
Customer Name: Manishwar
Customer ID: 1276
Billing Month: August 2024
Bill Date: 22-08-2024

CONSUMPTION DETAILS
=====
Units Consumed (kWh): 120.00
Rate per Unit: 6.50
=====
BILL BREAKDOWN
=====
1. Energy Charges (EC) ₹780.00
(120.00 units × ₹6.50)
2. Fixe Charges (FC) ₹50.00
3. Customer Charges (CC) ₹30.00
4. Electricity Duty (ED) ₹129.00
(15% of EC + FC + CC)
=====
TOTAL BILL AMOUNT ₹989.00
=====
CALCULATION SUMMARY
=====
Total Bill = EC + FC + CC + ED
Total Bill = ₹780.00 + ₹50.00 + ₹30.00 + ₹129.00
Total Bill = ₹989.00
=====

Do you want to calculate another bill? (y/n): y
Enter your choice (1-4): 2

```

The screenshot shows a Jupyter Notebook environment with several cells and a terminal tab.

**Code Cells:**

- Cell 1:** Electricity bill generation and analysis. It includes a table for "BILL BREAKDOWN" and a "CALCULATION SUMMARY".
- Cell 2:** Task 5: Final Bill Generation and Output Analysis. It includes a table for "ELECTRICITY BILL" and "CONSUMPTION DETAILS".
- Cell 3:** Task 5: Final Bill Generation and Output Analysis. It includes a table for "BILL BREAKDOWN" and a "CALCULATION SUMMARY".

**Terminal Tab:**

```
(.venv) (base) bodla.manishwar@BodlaManishwar-Laptop AIC % "/Users/bodla.manishwar/Downloads/AI Assistant coding/AIC/.venv/bin/python" "/User
```

**File Explorer:**

- AIC
- Python
- sample\_data.csv
- sample\_output.txt
- Task2\_File\_Operatio
- Task3\_CSV\_Data\_A
- Task5\_Final\_Bill\_Ge

## **CODE EXPLANATION:**

This program generates a complete electricity bill by accurately calculating Energy Charges (EC), Fixed Charges (FC), Customer Charges (CC), and Electricity Duty (ED), and then computing the total bill as the sum of all components. It presents the bill in a neatly formatted, professional layout with clear sectioning for consumption details and charge breakdown.