ASSINMENT 3.3

NAME:K.HITHESH

HALLTICKET:2303A51291

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

An electricity billing system must collect accurate consumer data.

Task Description

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

• Reads:

o Previous Units (PU)

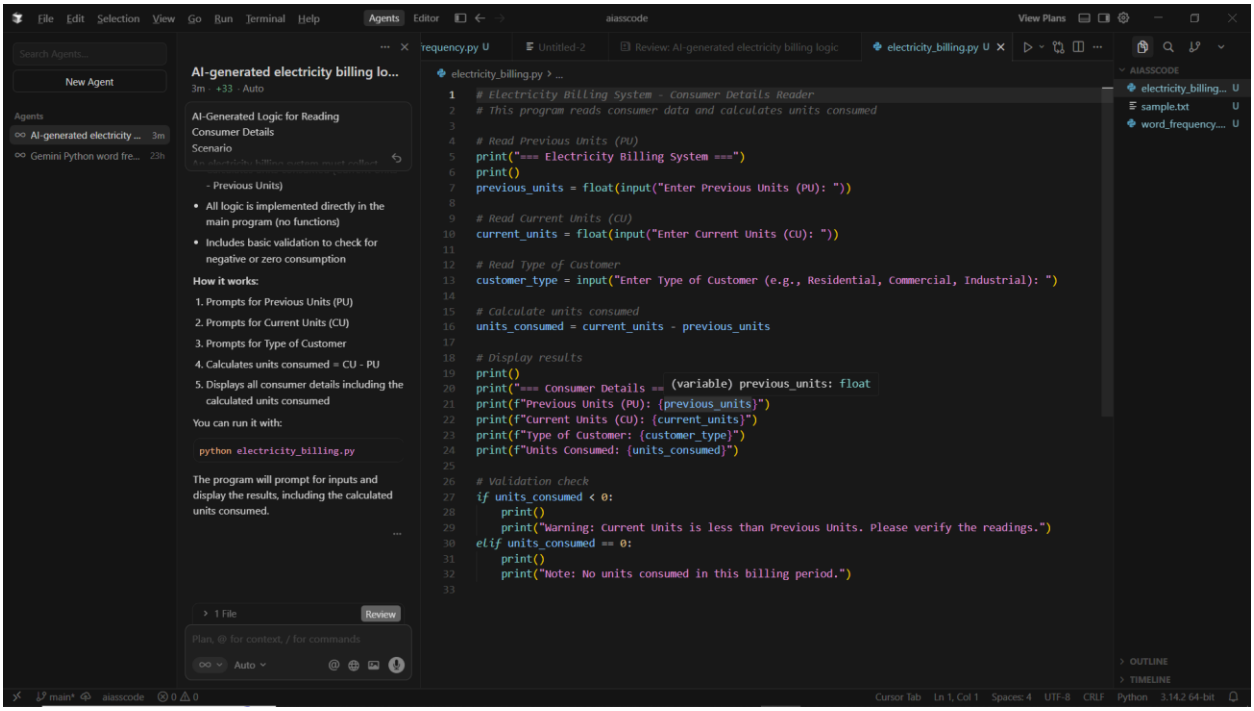
o Current Units (CU)

o Type of Customer

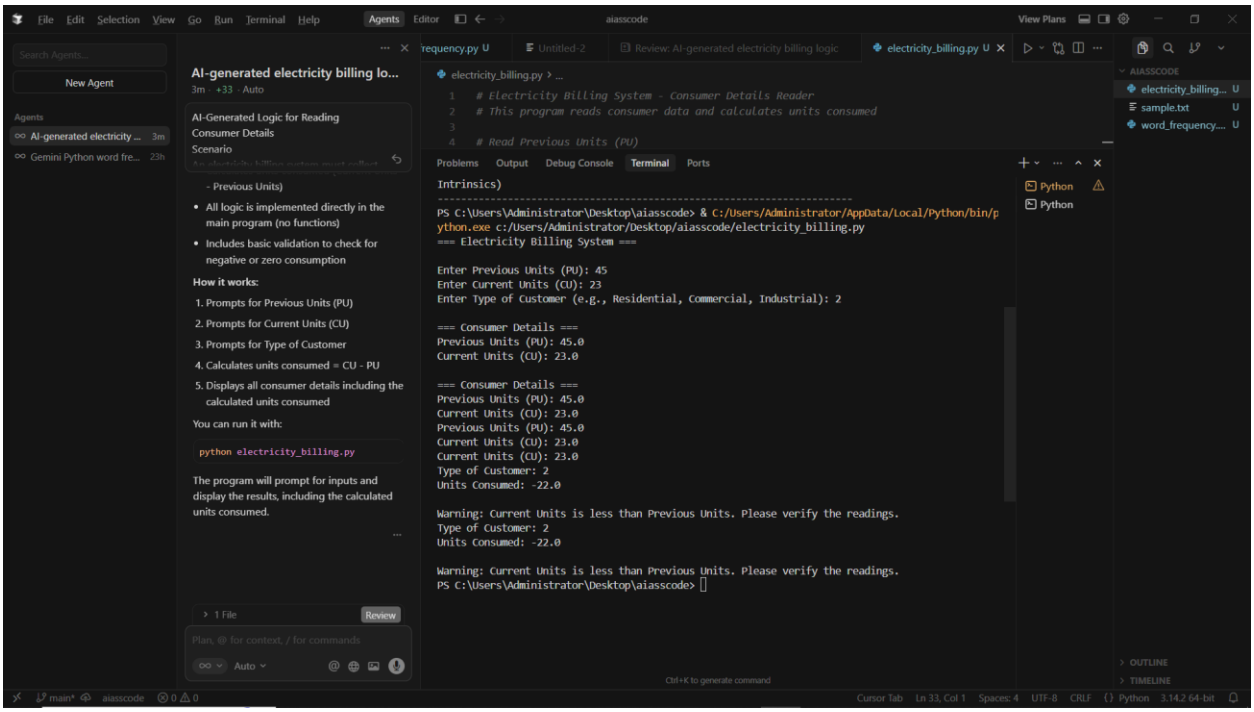
• Calculates units consumed

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

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



OUTPUT:



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

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

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

Task Description

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

• Calculate Energy Charges (EC)

• Use conditional statements based on:

o Domestic

o Commercial

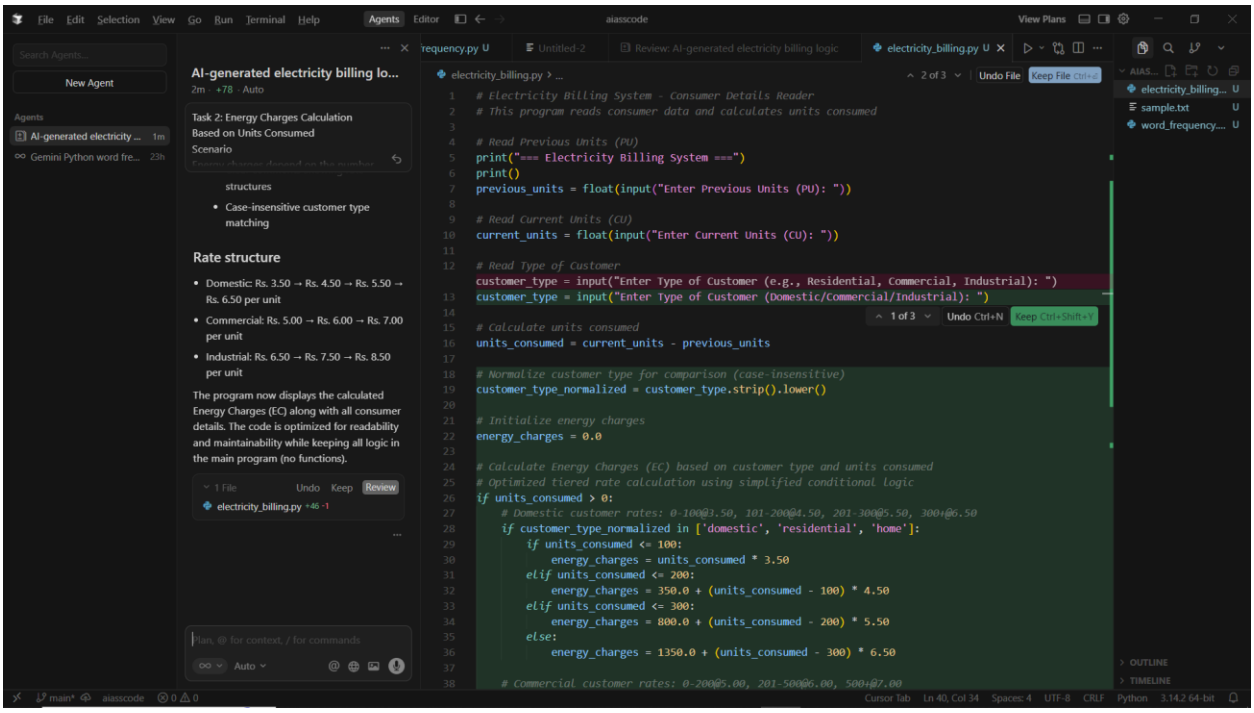
o Industrial consumers

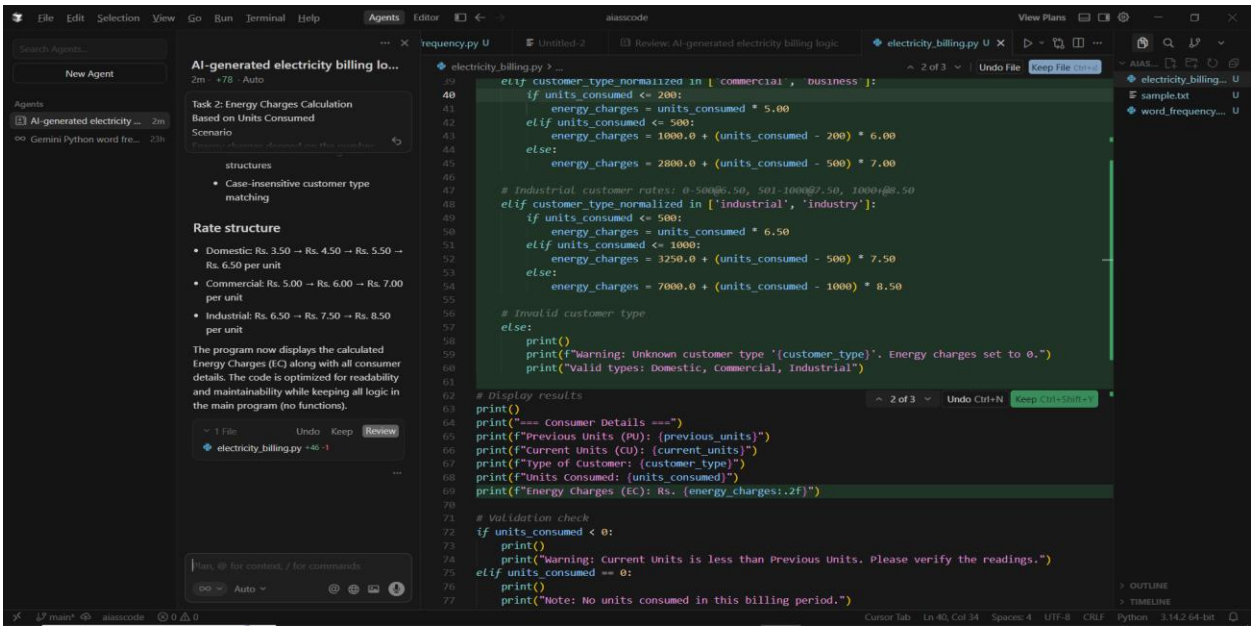
• Improve readability using AI prompts such as:

o “Simplify energy charge calculation logic”

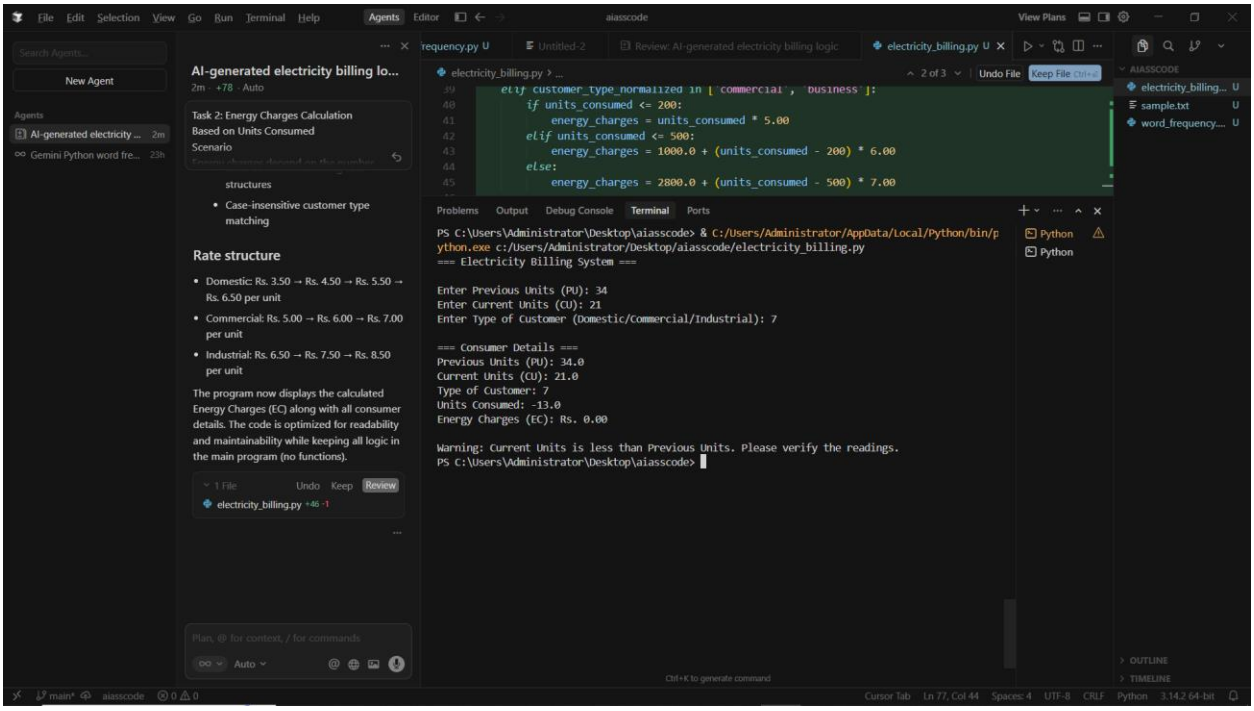
o “Optimize conditional statements”

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





OUTPUT:



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:

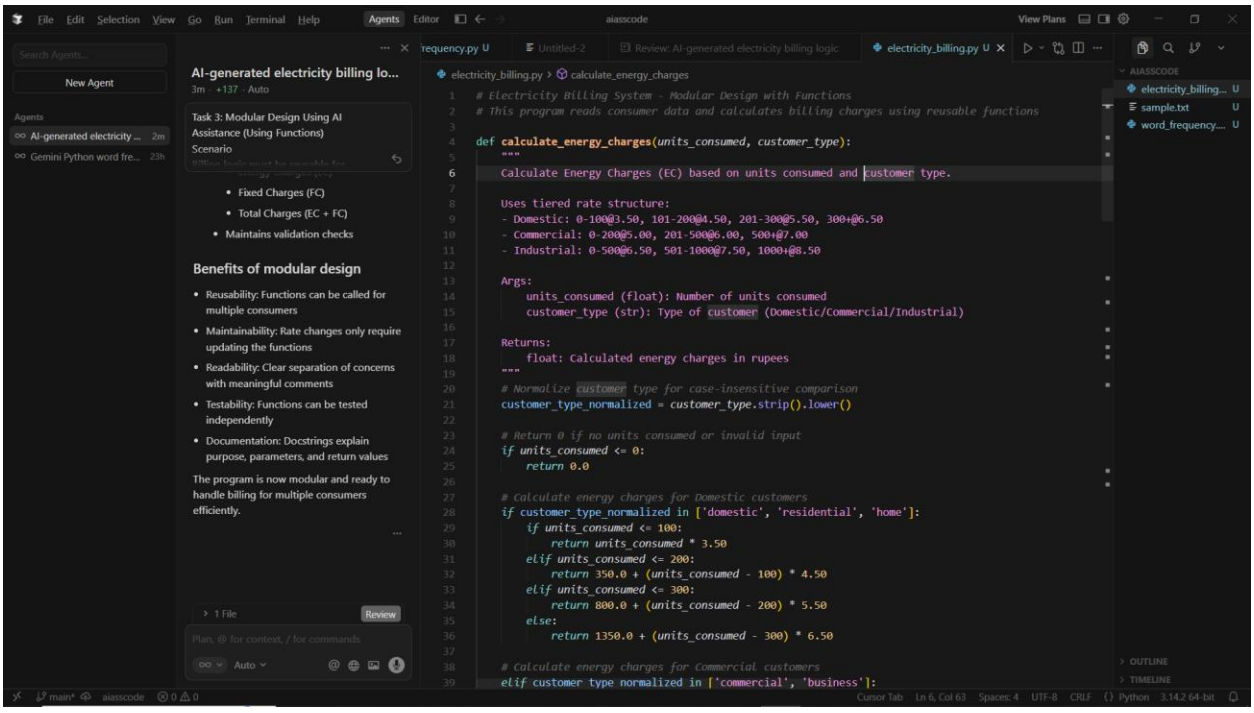
• Uses user-defined functions to:

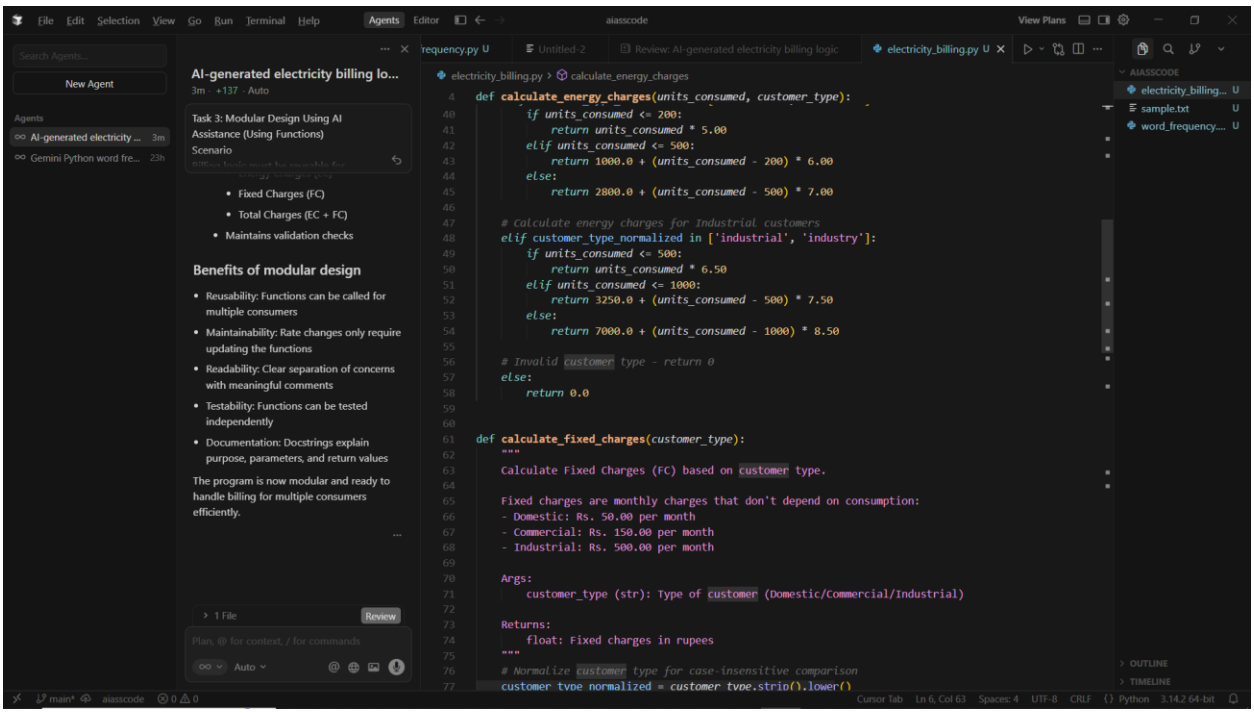
o Calculate Energy Charges

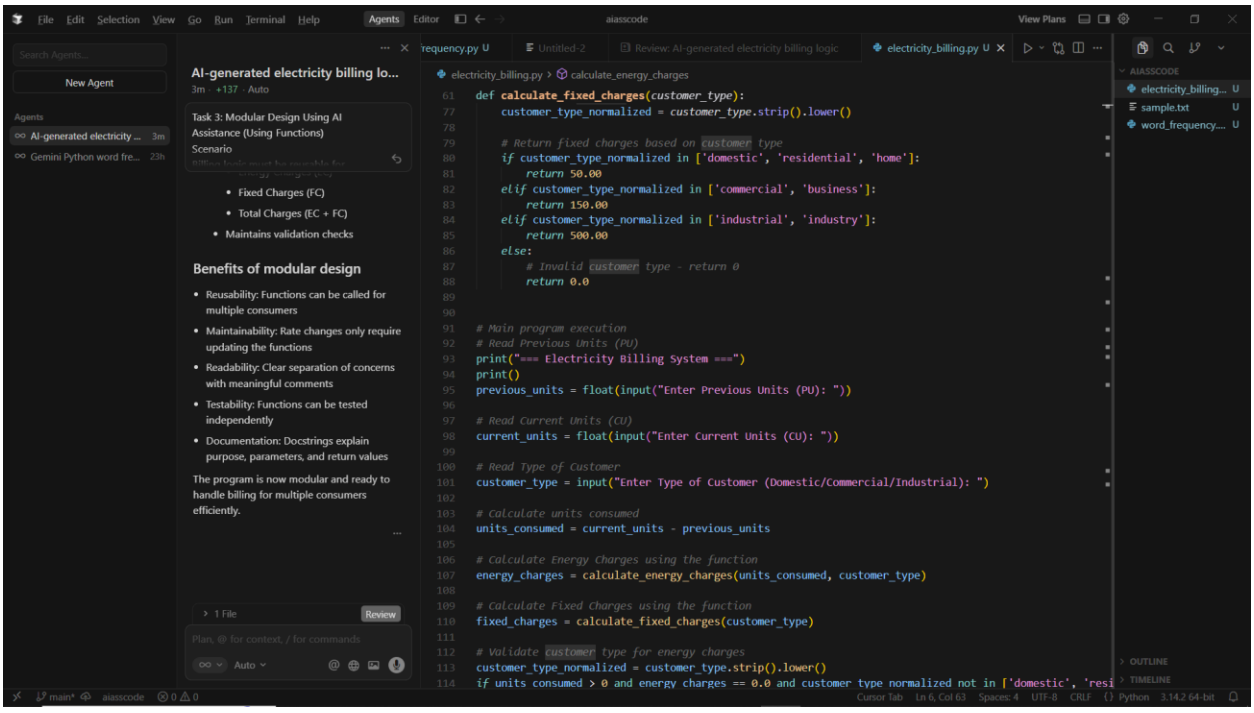
o Calculate Fixed Charges

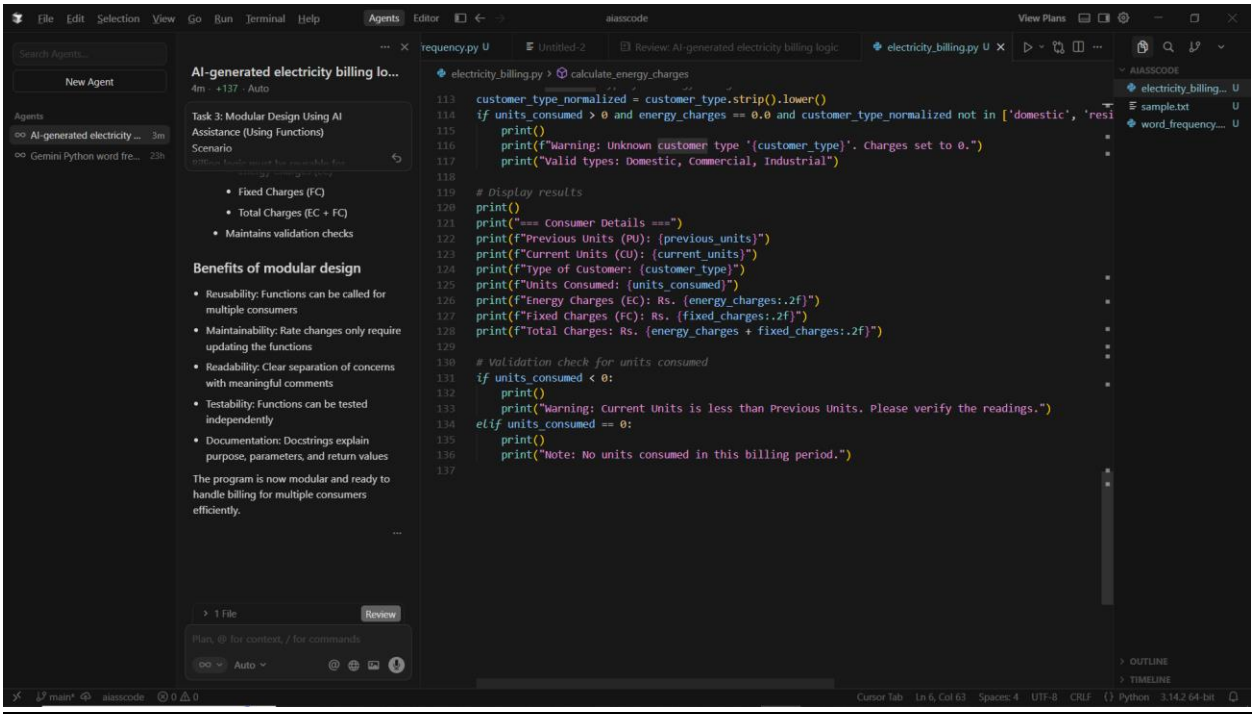
• Returns calculated values

• Includes meaningful comments

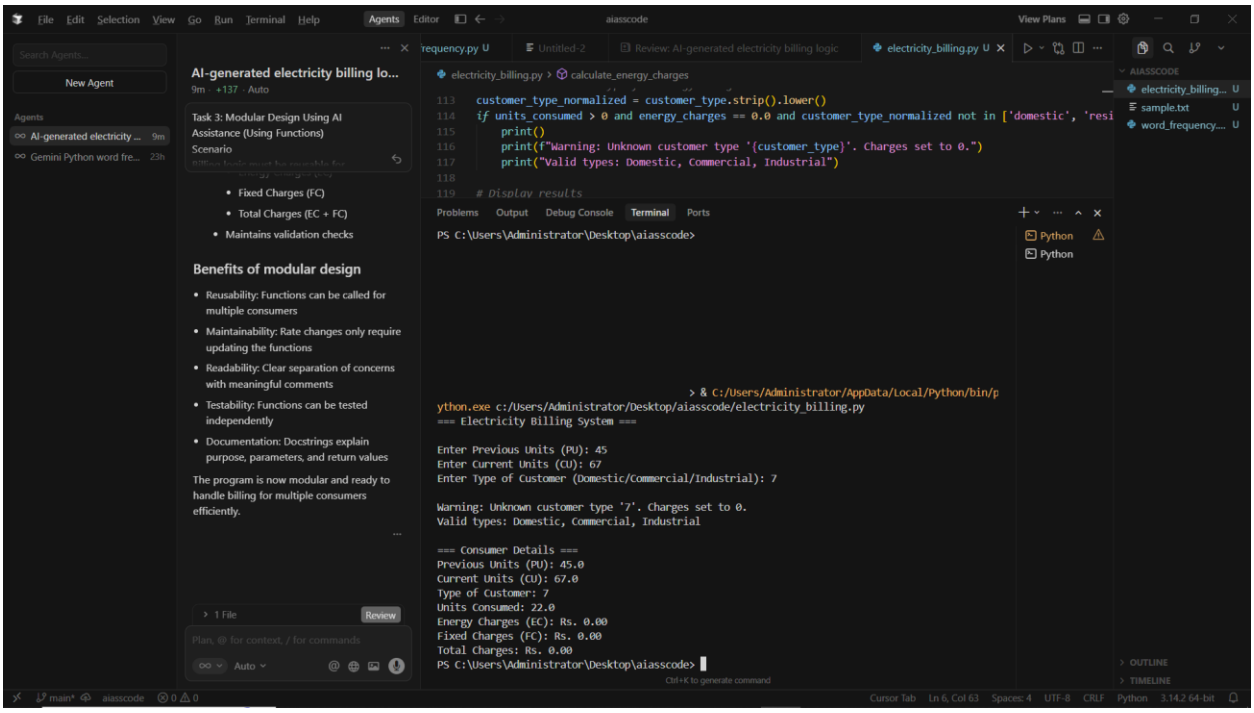

File Edit Selection View Go Run Terminal Help
Agents
Editor
aiasscode
View Plans
New Agent
Agents
AI-generated electricity ... 2m
Gemini Python word fre... 23h
AI-generated electricity billing lo...
3m · +137 · Auto
Task 3: Modular Design Using AI Assistance (Using Functions)
Scenario
Fixed Charges (FC)
Total Charges (EC + FC)
Maintains validation checks
Benefits of modular design
Reusability: Functions can be called for multiple consumers
Maintainability: Rate changes only require updating the functions
Readability: Clear separation of concerns with meaningful comments
Testability: Functions can be tested independently
Documentation: Docstrings explain purpose, parameters, and return values
The program is now modular and ready to handle billing for multiple consumers efficiently.
1 File
Review
Plan, @ for context, / for commands
Auto
Untitled-2
Review: AI-generated electricity billing logic
electricity_billing.py U
electricity_billing.py > calculate_energy_charges
# Electricity Billing System - Modular Design with Functions
# This program reads consumer data and calculates billing charges using reusable functions
def calculate_energy_charges(units_consumed, customer_type):
"""
Calculate Energy Charges (EC) based on units consumed and customer type.
Uses tiered rate structure:
- Domestic: 0-100@3.50, 101-200@4.50, 201-300@5.50, 300+@6.50
- Commercial: 0-200@5.00, 201-500@6.00, 500+@7.00
- Industrial: 0-500@6.50, 501-1000@7.50, 1000+@8.50
Args:
units_consumed (float): Number of units consumed
customer_type (str): Type of customer (Domestic/Commercial/Industrial)
Returns:
float: Calculated energy charges in rupees
"""
# Normalize customer type for case-insensitive comparison
customer_type_normalized = customer_type.strip().lower()
# Return 0 if no units consumed or invalid input
if units_consumed <= 0:
return 0.0
# Calculate energy charges for Domestic customers
if customer_type_normalized 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
# Calculate energy charges for Commercial customers
elif customer_type_normalized in ['commercial', 'business']:
AIASSCODE
electricity_billing... U
sample.txt U
word_frequency.... U
OUTLINE
TIMELINE
main* aiasscode 0 0
Cursor Tab Ln 6, Col 63 Spaces: 4 UTF-8 CRLF Python 3.14.2 64-bit


File Edit Selection View Go Run Terminal Help
Agents
Editor
aiasscode
View Plans
New Agent
Agents
AI-generated electricity ... 3m
Gemini Python word fre... 23h
AI-generated electricity billing lo...
3m · +137 · Auto
Task 3: Modular Design Using AI Assistance (Using Functions)
Scenario
Fixed Charges (FC)
Total Charges (EC + FC)
Maintains validation checks
Benefits of modular design
Reusability: Functions can be called for multiple consumers
Maintainability: Rate changes only require updating the functions
Readability: Clear separation of concerns with meaningful comments
Testability: Functions can be tested independently
Documentation: Docstrings explain purpose, parameters, and return values
The program is now modular and ready to handle billing for multiple consumers efficiently.
1 File
Review
Plan, @ for context, / for commands
Auto
Untitled-2
Review: AI-generated electricity billing logic
electricity_billing.py U
electricity_billing.py > calculate_energy_charges
def calculate_energy_charges(units_consumed, customer_type):
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
# Calculate energy charges for Industrial customers
elif customer_type_normalized 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
# Invalid customer type - return 0
else:
return 0.0
def calculate_fixed_charges(customer_type):
"""
Calculate Fixed Charges (FC) based on customer type.
Fixed charges are monthly charges that don't depend on consumption:
- Domestic: Rs. 50.00 per month
- Commercial: Rs. 150.00 per month
- Industrial: Rs. 500.00 per month
Args:
customer_type (str): Type of customer (Domestic/Commercial/Industrial)
Returns:
float: Fixed charges in rupees
"""
# Normalize customer type for case-insensitive comparison
customer_type_normalized = customer_type.strip().lower()
AIASSCODE
electricity_billing... U
sample.txt U
word_frequency.... U
OUTLINE
TIMELINE
main* aiasscode 0 0
Cursor Tab Ln 6, Col 63 Spaces: 4 UTF-8 CRLF Python 3.14.2 64-bit


AI-generated electricity billing lo...
3m · +137 · Auto
Task 3: Modular Design Using AI Assistance (Using Functions)
Scenario
Fixed Charges (FC)
Total Charges (EC + FC)
Maintains validation checks
Benefits of modular design
Reusability: Functions can be called for multiple consumers
Maintainability: Rate changes only require updating the functions
Readability: Clear separation of concerns with meaningful comments
Testability: Functions can be tested independently
Documentation: Docstrings explain purpose, parameters, and return values
The program is now modular and ready to handle billing for multiple consumers efficiently.
1 File
Review
Plan, @ for context, / for commands
Auto
def calculate_fixed_charges(customer_type):
customer_type_normalized = customer_type.strip().lower()
# Return fixed charges based on customer type
if customer_type_normalized in ['domestic', 'residential', 'home']:
return 50.00
elif customer_type_normalized in ['commercial', 'business']:
return 150.00
elif customer_type_normalized in ['industrial', 'industry']:
return 500.00
else:
# Invalid customer type - return 0
return 0.0
# Main program execution
# 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 (Domestic/Commercial/Industrial): ")
# Calculate units consumed
units_consumed = current_units - previous_units
# Calculate Energy Charges using the function
energy_charges = calculate_energy_charges(units_consumed, customer_type)
# Calculate Fixed Charges using the function
fixed_charges = calculate_fixed_charges(customer_type)
# Validate customer type for energy charges
customer_type_normalized = customer_type.strip().lower()
if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'resi


AI-generated electricity billing lo...
4m · +137 · Auto
Task 3: Modular Design Using AI Assistance (Using Functions)
Scenario
Fixed Charges (FC)
Total Charges (EC + FC)
Maintains validation checks
Benefits of modular design
Reusability: Functions can be called for multiple consumers
Maintainability: Rate changes only require updating the functions
Readability: Clear separation of concerns with meaningful comments
Testability: Functions can be tested independently
Documentation: Docstrings explain purpose, parameters, and return values
The program is now modular and ready to handle billing for multiple consumers efficiently.
1 File
Review
Plan, @ for context, / for commands
Auto
customer_type_normalized = customer_type.strip().lower()
if units_consumed > 0 and energy_charges == 0.0 and customer_type_normalized not in ['domestic', 'resi
print()
print(f"Warning: Unknown customer type '{customer_type}'. Charges set to 0.")
print("Valid types: Domestic, Commercial, Industrial")
# 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}")
print(f"Energy Charges (EC): Rs. {energy_charges:.2f}")
print(f"Fixed Charges (FC): Rs. {fixed_charges:.2f}")
print(f"Total Charges: Rs. {energy_charges + fixed_charges:.2f}")
# Validation check for units consumed
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:



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)

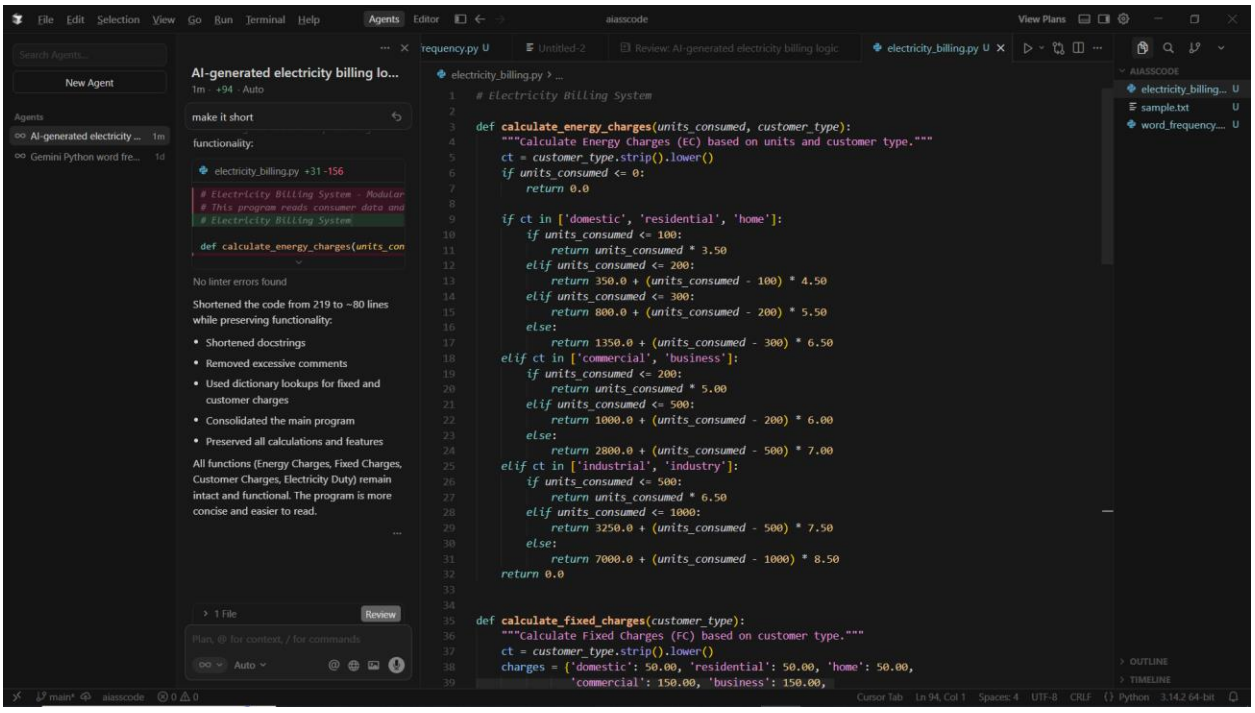
Use AI prompts like:

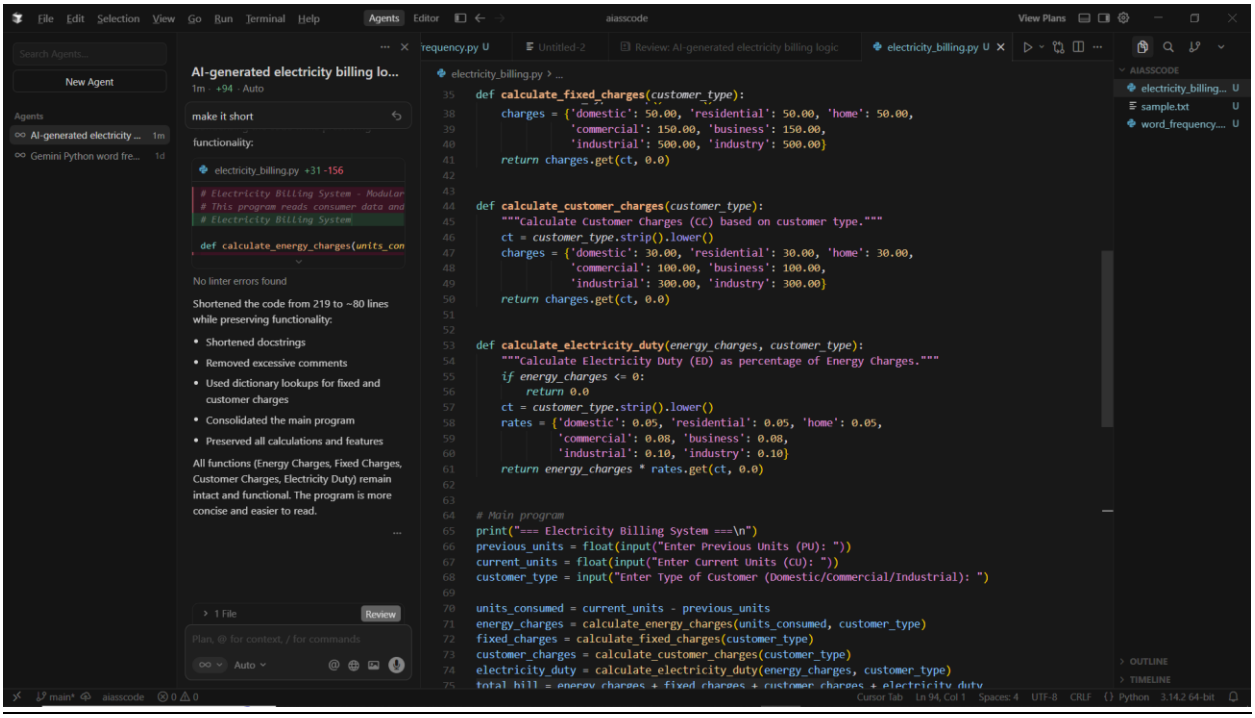
• "Add electricity duty calculation"

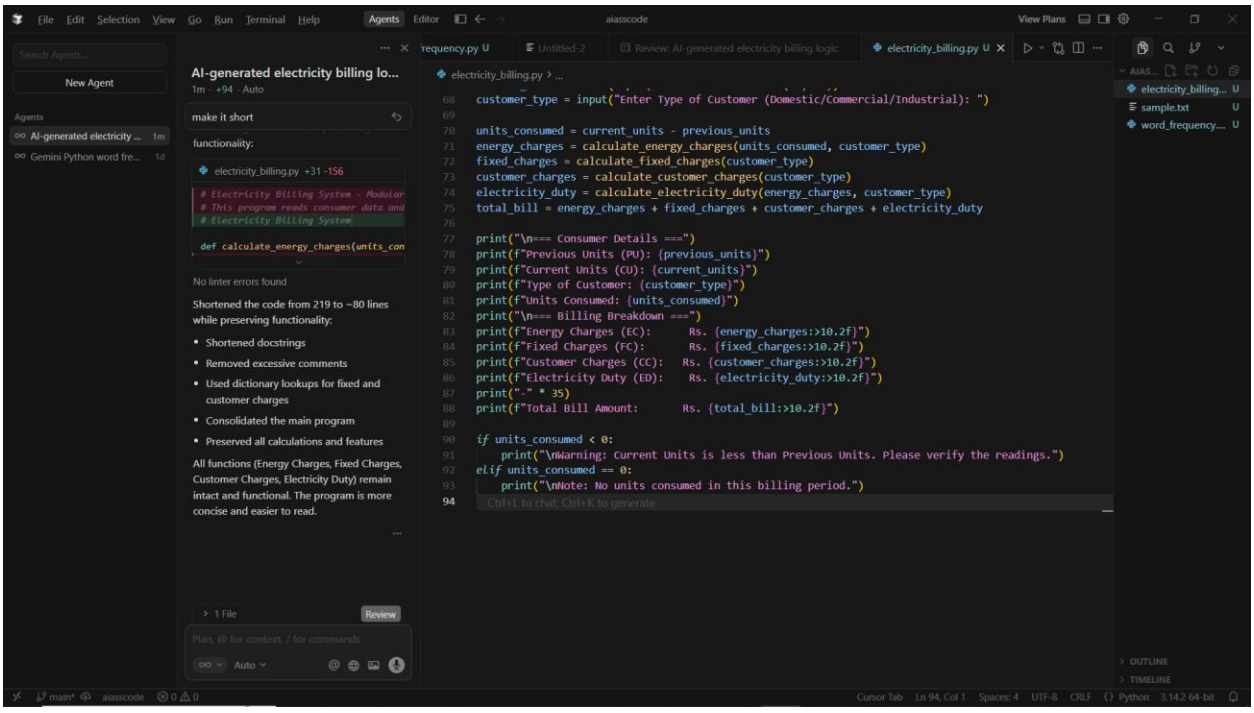
• "Improve billing accuracy"

PROMPT:

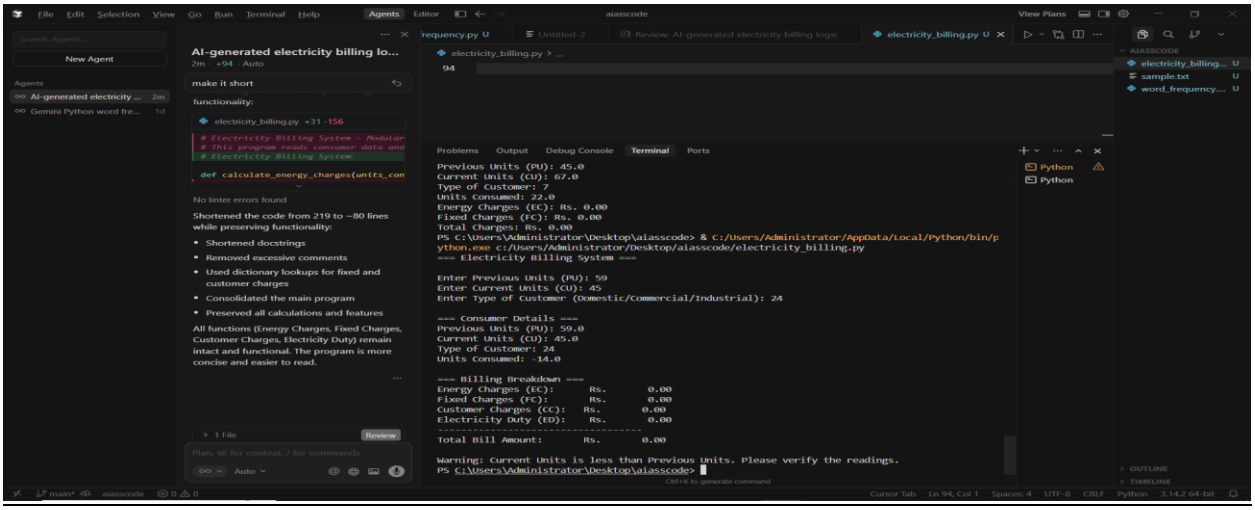
Write a program to calculate and display an electricity bill using meter readings, customer type, and applicable charges


File Edit Selection View Go Run Terminal Help
Agents
Editor
aiasscode
View Plans
New Agent
Agents
AI-generated electricity ... 1m
Gemini Python word fre... 1d
AI-generated electricity billing lo...
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_con
No linter errors found
Shortened the code from 219 to ~80 lines while preserving functionality:
Shortened docstrings
Removed excessive comments
Used dictionary lookups for fixed and customer charges
Consolidated the main program
Preserved all calculations and features
All functions (Energy Charges, Fixed Charges, Customer Charges, Electricity Duty) remain intact and functional. The program is more concise and easier to read.
1 File
Review
Plan, @ for context, / for commands
Auto
electricity_billing.py U
Untitled-2
Review: AI-generated electricity billing logic
electricity_billing.py
# Electricity Billing System
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."""
ct = customer_type.strip().lower()
charges = {'domestic': 50.00, 'residential': 50.00, 'home': 50.00,
'commercial': 150.00, 'business': 150.00,
AIASSCODE
electricity_billing... U
sample.txt U
word_frequency.... U
OUTLINE
TIMELINE
main* aiasscode 0 0
Cursor Tab Ln 94, Col 1 Spaces: 4 UTF-8 CRLF Python 3.14.2 64-bit


File Edit Selection View Go Run Terminal Help
Agents
Editor
aiasscode
View Plans
New Agent
Agents
AI-generated electricity ... 1m
Gemini Python word fre... 1d
AI-generated electricity billing lo...
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_con
No linter errors found
Shortened the code from 219 to ~80 lines while preserving functionality:
Shortened docstrings
Removed excessive comments
Used dictionary lookups for fixed and customer charges
Consolidated the main program
Preserved all calculations and features
All functions (Energy Charges, Fixed Charges, Customer Charges, Electricity Duty) remain intact and functional. The program is more concise and easier to read.
1 File
Review
Plan, @ for context, / for commands
Auto
electricity_billing.py U
Untitled-2
Review: AI-generated electricity billing logic
electricity_billing.py
def calculate_fixed_charges(customer_type):
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
AIASSCODE
electricity_billing... U
sample.txt U
word_frequency.... U
OUTLINE
TIMELINE
main* aiasscode 0 0
Cursor Tab Ln 94, Col 1 Spaces: 4 UTF-8 CRLF Python 3.14.2 64-bit



OUTPUT:



EXPLANATION:

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

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

Task Description

Develop the final Python application to:

• Calculate total bill:

• Total Bill = EC + FC + CC + ED

• Display:

o Energy Charges (EC)

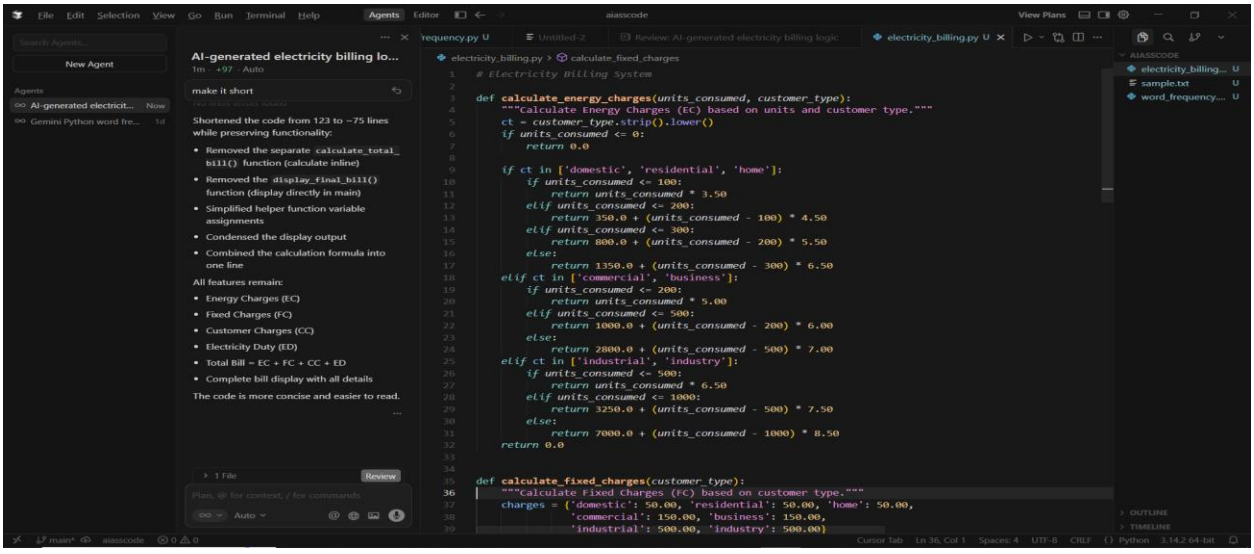
o Fixed Charges (FC)

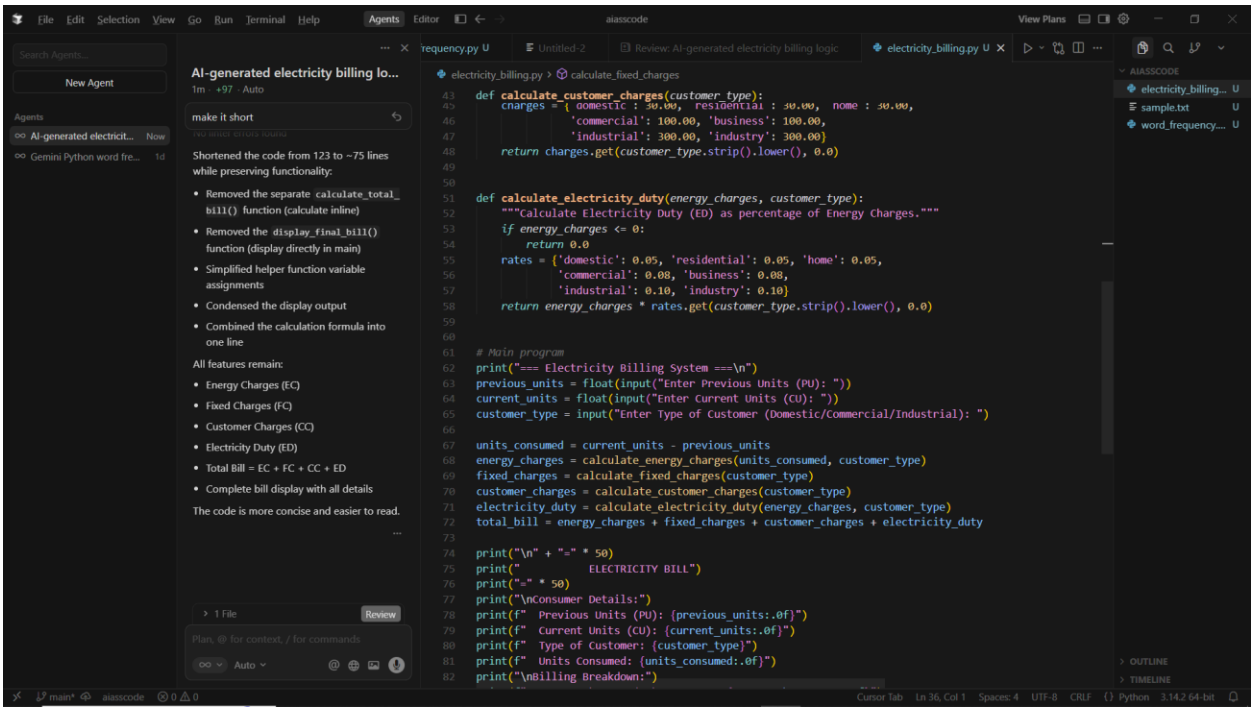
o Customer Charges (CC)

o Electricity Duty (ED)

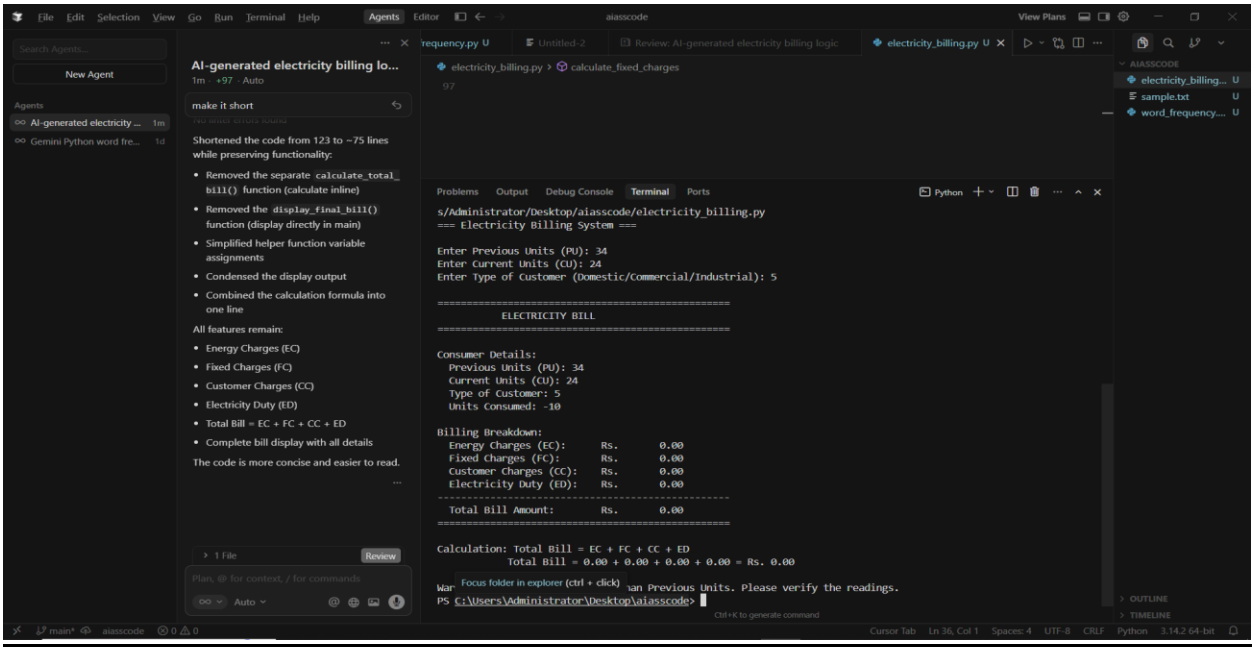
o Total Bill Amount

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





OUTPUT



EXPLANATION:

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