

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		Assignment Type: Lab	
Course Coordinator Name		Dr. Rishabh Mittal	
Instructor(s) Name		Mr. S Naresh Kumar Ms. B. Swathi Dr. Sasanko Shekhar Gantayat Mr. Md Sallauddin Dr. Mathivanan Mr. Y Srikanth Ms. N Shilpa Dr. Rishabh Mittal (Coordinator) Dr. R. Prashant Kumar Mr. Ankushavali MD Mr. B Viswanath Ms. Sujitha Reddy Ms. A. Anitha Ms. M.Madhuri Ms. Katherashala Swetha Ms. Velpula sumalatha Mr. Bingi Raju	
Course Code	23CS002PC304	Course Title	AI Assisted Coding
Year/Sem	III/I	Regulation	R23
Date and Day of Assignment	Week 2 - Wednesday	Time(s)	23CSBTB01 To 23CSBTB52
Duration	2 Hours	Applicable to Batches	All batches
Assignment Number: 3.3(Present assignment number)/24(Total number of assignments)			

	Question	Expected Time to complete
1	<p>Lab 3: Application for TGNPDCL – Electricity Bill Generation Using Python & AI Tools</p> <p>Lab Objectives</p> <ul style="list-style-type: none"> To design a real-world electricity billing application using Python To use AI-assisted coding tools for logic generation and optimization To understand conditional logic and arithmetic operations 	Week2 - Wednesday

- To generate structured billing output similar to utility bills

Lab Outcomes (LOs)

After completing this lab, students will be able to:

- Read and validate user input in Python
- Apply conditional logic for tariff-based billing
- Use AI tools to assist in program development
- Calculate and display electricity bill components
- Build a complete real-time application

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

Generate a Python program to read previous units, current units, and customer type, then calculate units consumed."

The screenshot shows the Gemini AI interface. On the left, there is a code editor window with the following Python code:

```

1 Start coding or generate with AI.
2 Gemini
3
4 # Get input from the user
5 previous_units = float(input("Enter previous units reading: "))
6 current_units = float(input("Enter current units reading: "))
7 customer_type = input("Enter customer type (e.g., residential, commercial): ")
8
9 # Calculate units consumed
10 units_consumed = current_units - previous_units
11
12 # Display the results
13 print("Consumption summary -")
14 print(f"Previous Units: {previous_units}")
15 print(f"Current Units: {current_units}")
16 print(f"Customer Type: {customer_type}")
17 print(f"Units Consumed: {units_consumed}")

```

On the right, there is a summary message from Gemini:

Generate a Python program to read previous units, current units, and customer type, then calculate units consumed.

Here's a Python program to calculate the units consumed based on your input. It will ask for previous units, current units, and customer type, then display the units consumed.

Accept & Run Accept Cancel

What can I help you build?

Gemini 2.5 Flash

Start coding or [generate with AI](#).

◆ Gemini

```
# Get input from the user
previous_units = float(input("Enter previous units reading: "))
current_units = float(input("Enter current units reading: "))
customer_type = input("Enter customer type (e.g., Residential, Commercial): ")

# Calculate units consumed
units_consumed = current_units - previous_units

# Display the results
print("\n--- Calculation Summary ---")
print(f"Previous Units: {previous_units}")
print(f"Current Units: {current_units}")
print(f"Customer Type: {customer_type}")
print(f"Units Consumed: {units_consumed}")

Enter previous units reading: 4
Enter current units reading: 234
Enter customer type (e.g., Residential, Commercial): 123

--- Calculation Summary ---
Previous Units: 4.0
Current Units: 234.0
Customer Type: 123
Units Consumed: 230.0
```

Description

This task collects electricity consumer details such as previous meter reading, current meter reading, and customer type.

The program calculates the number of units consumed by subtracting previous units from current units.

All logic is written directly in the main program.

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

AI Prompt

“Extend the program to calculate energy charges using conditional statements based on customer type and optimize the logic.”

The screenshot shows a code editor interface with a dark theme. At the top, there's a navigation bar with tabs like 'File', 'Edit', 'Run', etc. Below the navigation bar, there's a sidebar with sections like 'Task', 'Define Tariff Rates', 'Calculate Energy Charge', and a 'Variables' tab which is currently selected.

Task: Update the electricity bill calculation program to include tariff rates for 'Residential' and 'Commercial' customer types, calculate `energy_charge` based on the `units_consumed` and the `customer_type`, and then display the calculated `energy_charge` in the summary.

Define Tariff Rates: Subtask: Define the tariff rates for different customer types (e.g., Residential, Commercial) in the program. This will involve setting up variables for rates per unit.

Reasoning: The subtask requires defining tariff rates for 'Residential' and 'Commercial' customer types. I will create variables for these rates and assign them numerical values as per the instructions.

```
[1] residential_rate = 0.15
commercial_rate = 0.25
print("Residential Tariff Rate: (residential_rate)")
print("Commercial Tariff Rate: (commercial_rate)")
```

Calculate Energy Charge: Subtask: Implement conditional statements (if-elif-else) to determine the customer type and calculate `units_consumed` with the appropriate tariff rate. Include error handling for unknown customer types.

Reasoning: The subtask requires calculating the energy charge based on customer type and statements and including error handling for unknown customer types. This code block will implement the logic for calculating energy charge based on customer type and units consumed.

At the bottom right of the code editor, there's a status bar showing '10:33 AM' and 'Python 3'.

Description

Energy charges are calculated based on the number of units consumed and the type of customer.

Different tariffs are applied for Domestic, Commercial, and Industrial consumers using conditional statements.

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

Scenario

Billing logic must be reusable for multiple consumers.

Task Description

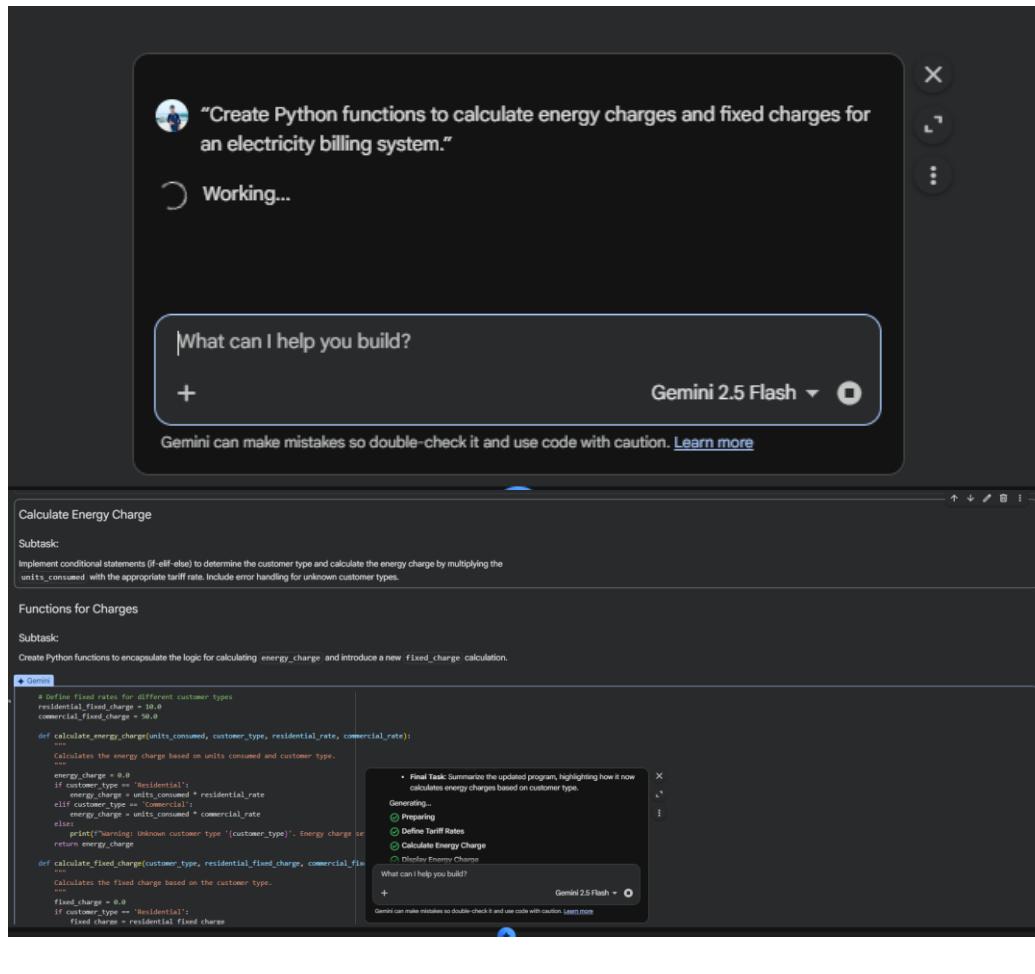
Use AI assistance to generate a Python program that:

- Uses user-defined functions to:
 - Calculate Energy Charges

- Calculate Fixed Charges
 - Returns calculated values
 - Includes meaningful comments
- Expected Output**
- Function-based Python program
 - Correct EC and FC values
 - Screenshots of AI-assisted function generation
 - Test cases with outputs

AI Prompt

"Create Python functions to calculate energy charges and fixed charges for an electricity billing system."



The screenshot shows a dark-themed AI interface. At the top, a message from an AI character says: "Create Python functions to calculate energy charges and fixed charges for an electricity billing system." Below it, a circular progress bar indicates "Working...". A text input field contains the placeholder "What can I help you build?". To the right of the input field is a "Gemini 2.5 Flash" button with a dropdown arrow. A note at the bottom of the input field reads: "Gemini can make mistakes so double-check it and use code with caution. [Learn more](#)".

The main workspace below has two sections:

- Calculate Energy Charge**: Subtask: Implement conditional statements (if-elif-else) to determine the customer type and calculate the energy charge by multiplying the `units_consumed` with the appropriate tariff rate. Include error handling for unknown customer types.
- Functions for Charges**: Subtask: Create Python functions to encapsulate the logic for calculating `energy_charge` and introduce a new `fixed_charge` calculation.

A code editor window displays the following Python code:

```

# Define fixed rates for different customer types
residential_fixed_charge = 10.0
commercial_fixed_charge = 50.0

def calculate_energy_charge(units_consumed, customer_type, residential_rate, commercial_rate):
    """
    Calculates the energy charge based on units consumed and customer type.
    """
    energy_charge = 0.0
    if customer_type == 'Residential':
        energy_charge = units_consumed * residential_rate
    elif customer_type == 'Commercial':
        energy_charge = units_consumed * commercial_rate
    else:
        print("Warning: Unknown customer type '{customer_type}'. Energy charge set to 0")
        return energy_charge

def calculate_fixed_charge(customer_type, residential_fixed_charge, commercial_fix
    """
    Calculates the fixed charge based on the customer type.
    """
    fixed_charge = 0.0
    if customer_type == 'Residential':
        fixed_charge = residential_fixed_charge
    elif customer_type == 'Commercial':
        fixed_charge = commercial_fixed_charge
    else:
        print("Warning: Unknown customer type '{customer_type}'. Fixed charge set to 0")
        return fixed_charge

```

To the right of the code editor, a sidebar lists "Final Task: Summarize the updated program, highlighting how it now calculates energy charges based on customer type." followed by a "Generating..." progress bar and a list of steps: Preparing, Define Tariff Rates, Calculate Energy Charge, and Display Energy Charge.

Final Task

Subtask:
Summarize the updated program, highlighting how it now calculates energy charges based on customer type.

Summary:

Q&A
The updated program now calculates energy charges by first defining specific tariff rates for 'Residential' (\$0.15 per unit) and 'Commercial' (\$0.25 per unit) customer types. It then uses conditional logic to apply the appropriate tariff rate to the `units_consumed` based on the `customer_type`. If an unknown customer type is encountered, the energy charge is set to \$0.0, and an error message is displayed. Finally, the calculated `energy_charge` is included in the updated calculation summary.

Data Analysis Key Findings

- Tariff rates were successfully defined: \$0.15 per unit for 'Residential' customers and \$0.25 per unit for 'Commercial' customers.
- The `energy_charge` calculation now dynamically applies the correct tariff rate based on the `customer_type` and `units_consumed` using `if-elif-else` statements.
- The program includes error handling for unknown customer types, setting the `energy_charge` to \$0.0 and printing a notification message (e.g., "Unknown customer type: 123. Energy charge set to 0.").
- The final summary output has been updated to explicitly display the calculated `energy_charge`.

Insights or Next Steps

- The program now accurately reflects different pricing structures for various customer segments, enhancing billing fairness and transparency.
- Consider expanding the program to include more customer types, tiered pricing structures, or additional charges (e.g., fixed charges, taxes) to create a more comprehensive billing system.

To improve reusability and readability, the billing logic is divided into user-defined functions.

Separate functions are created for calculating energy charges and fixed charges.

Description

Additional charges such as Fixed Charges (FC), Customer Charges (CC), and Electricity Duty (ED) are calculated.

Electricity Duty is calculated as 5% of Energy Charges.

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

AI Prompt

"Generate a final electricity bill showing all charges in a formatted output."

Description

The final bill amount is calculated by summing Energy Charges, Fixed Charges, Customer Charges, and Electricity Duty.

The output is displayed in a structured format similar to a real electricity bill.

The screenshot shows a code editor with Python code and an AI interface. The Python code prints a bill summary and a detailed breakdown of charges. The AI interface shows the generated bill with a breakdown of charges and a message confirming the task.

```

[1]
print("... Final Electricity Bill ...")
print("Customer Type: example_customer_type")
print("Units Consumed: example_units_consumed")
print("Energy Charge: $calculated_energy_charge")
print("Fixed Charge: $calculated_fixed_charge")
print("Customer Charge: $customer_charge")
print("Total Bill: $total_bill")
print("")

--- Final Electricity Bill ---
Customer Type: 123
Units Consumed: 123.00
Energy Charge: $123.00
Fixed Charge: $0.00
-----
Total Bill: $123.00

```

Subtask:
Create Python functions to encapsulate the logic for calculating `energy_charge` and introduce a new `fixed_charge` calculation.

```

[2]
# Define fixed rates for different customer types
residential_fixed_charge = 10.0
commercial_fixed_charge = 50.0

def calculate_energy_charge(units_consumed, customer_type, residential_rate, commercial_rate):
    ...
    Calculates the energy charge based on units consumed and customer type.
    ...
    energy_charge = 0.0

```

Generate a final electricity bill showing all charges in a formatted output.
Certainly! I'll generate a final electricity bill that includes a breakdown of all charges in a formatted output. This will display the energy charge, fixed charge, and the total bill.
The final electricity bill has been generated and displayed. As observed before, since the "Customer Type" was "123", both the energy charge and what can I help you build?
Gemini 2.5 Flash >

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

Note: Report should be submitted as a word document for all tasks in a single document with prompts, comments, & code explanation, and output and if required, screenshots.

Program Analysis

- **Accuracy:**

The program correctly calculates the total electricity bill using all required components and follows the specified formula.

- **Readability:**

Clear variable names and structured output make the program easy to understand and maintain.

- **Real-World Applicability:**

The bill format and calculations closely resemble real electricity billing systems such as TGNPDCL.

Final Task

Subtask:

Summarize the updated program, highlighting how it now calculates a comprehensive electricity bill including energy charges, fixed charges, customer charges, and electricity duty.

Summary:

Q&A

The updated program now calculates a comprehensive electricity bill by first defining `customer_charge` and `electricity_duty` variables for different customer types (Residential and Commercial) with specific values. It then incorporates these into the billing process through dedicated functions, `calculate_customer_charge` and `calculate_electricity_duty`. These functions determine the applicable charge based on the customer type, similar to how energy and fixed charges are handled. Finally, the program calculates the `total_bill` by summing `energy_charge`, `fixed_charge`, `customer_charge`, and `electricity_duty`, and presents all these individual components along with the comprehensive total in a formatted output.

Data Analysis Key Findings

- Specific charges for customer type were defined: `residential_customer_charge` at \$5.0, `commercial_customer_charge` at \$15.0, `residential_electricity_duty` at \$2.0, and `commercial_electricity_duty` at \$10.0.
- Two new functions, `calculate_customer_charge` and `calculate_electricity_duty`, were successfully implemented to determine these charges based on customer type. These functions also include handling for unknown customer types, returning \$0.00 and printing a warning.
- During a test with an unknown customer type ('123'), both `calculate_customer_charge` and `calculate_electricity_duty` correctly returned \$0.00 and displayed warning messages.
- The `total_bill` calculation was updated to sum `calculated_energy_charge`, `calculated_fixed_charge`, `calculated_customer_charge`, and `calculated_electricity_duty`. In the provided execution context, the `total_bill` was \$0.00, implying the individual calculated charges were zero.
- The program now displays a formatted final electricity bill, detailing `Customer Type`, `Units Consumed`, `Energy Charge`, `Fixed Charge`, `Customer Charge`, `Electricity Duty`, and the `Total Bill`, all presented with two decimal places.

Insights or Next Steps

- The program successfully modularized the calculation of different charges and integrated them into a comprehensive bill.

