AI ASSISTANT CODING

2303A51395

Batch – 06

Assignment – 3.3

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)

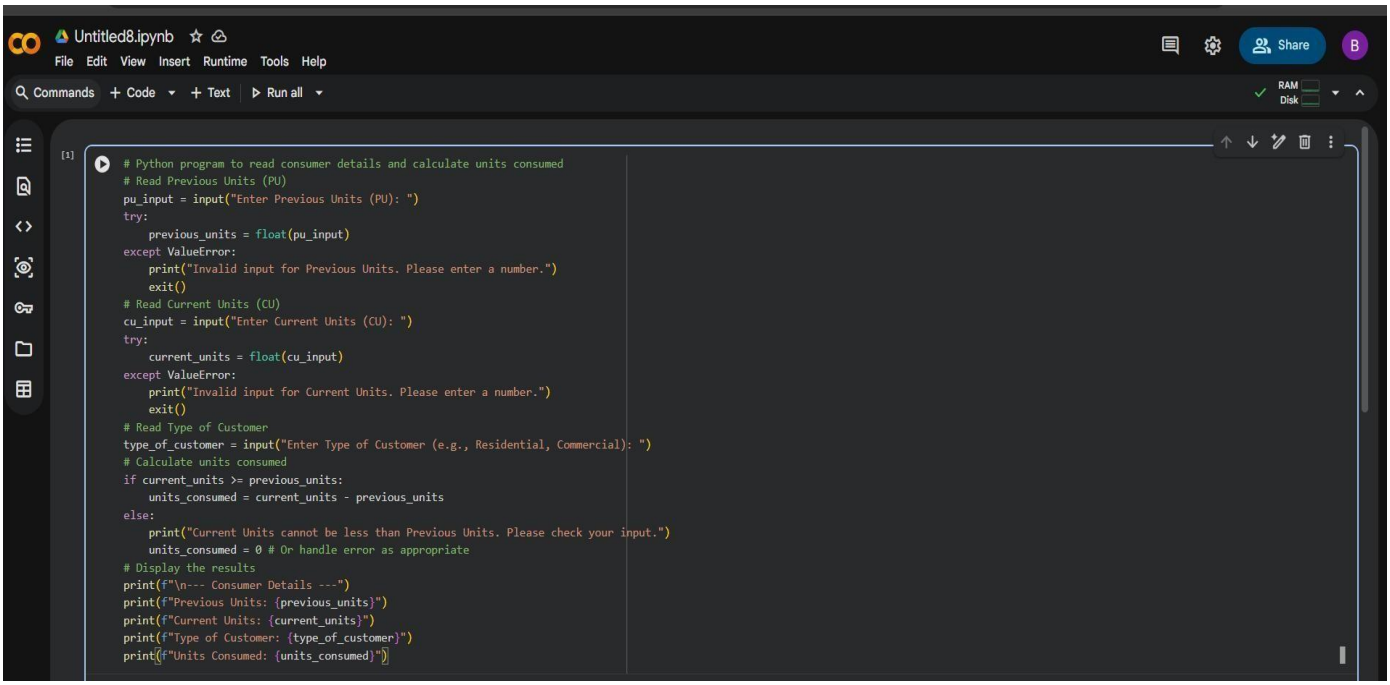
Expected Output

• Correct input reading

• Units consumed calculation

• Screenshot showing AI-generated code

• Sample input and output



Output:-

Enter Previous Units (PU): 4

Enter Current Units (CU): 3

Enter Type of Customer (e.g., Residential, Commercial): Residential

Current Units cannot be less than Previous Units. Please check your input.

--- Consumer Details ---

Previous Units: 4.0

Current Units: 3.0

Type of Customer: Residential

Units Consumed: 0

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”

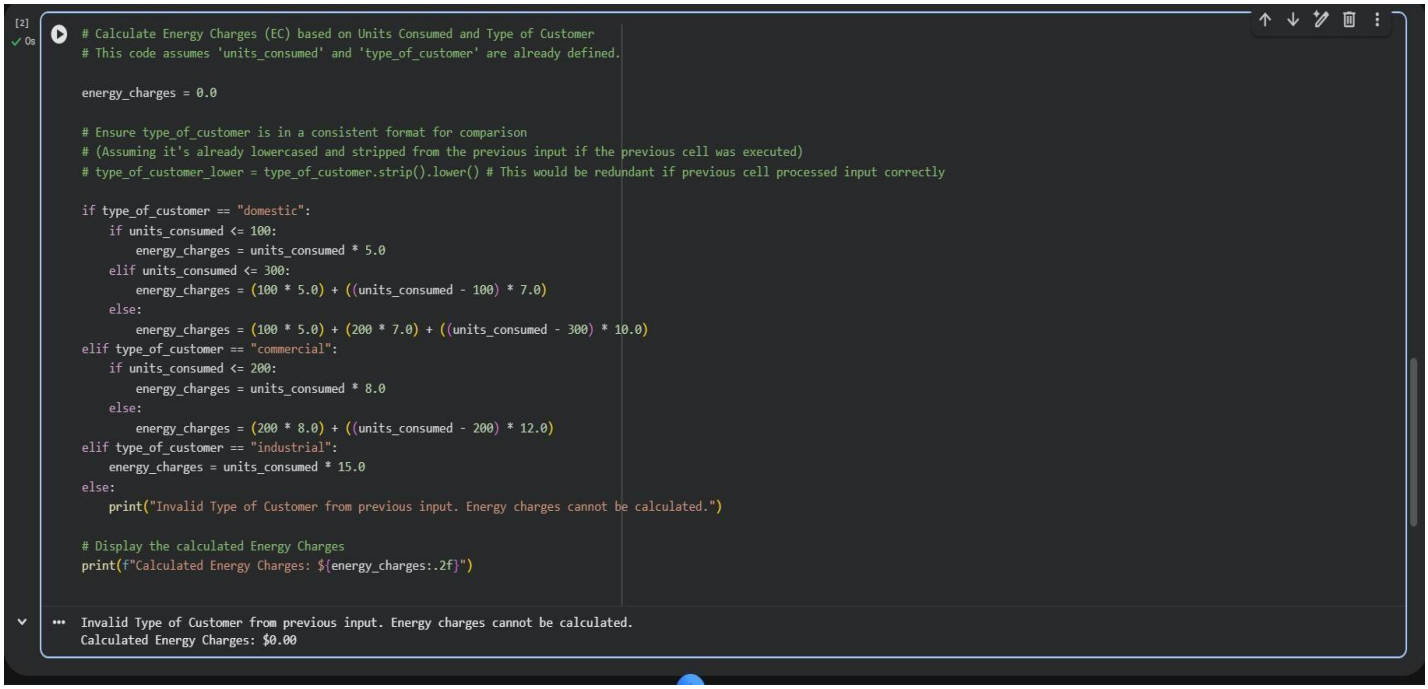
Expected Output

• Correct EC calculation

• Clear conditional logic

• Original and improved versions (optional)

• Sample execution results



Output:-

Calculated Energy Charges: $400.00

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

Expected Output

• Function-based Python program

• Correct EC and FC values

• Screenshots of AI-assisted function generation

• Test cases with outputs

Code:-

```
# Function to calculate Energy Charges based on units consumed and customer type
def calculate_energy_charges(units_consumed, type_of_customer):
energy_charges = 0.0
    # Convert type_of_customer to lowercase for consistent comparison
    type_of_customer_lower = type_of_customer.strip().lower()

    if type_of_customer_lower == "domestic":
        if units_consumed <= 100:
            energy_charges = units_consumed * 5.0
        elif units_consumed <= 300:
            energy_charges = (100 * 5.0) + ((units_consumed - 100) * 7.0)
        else:
```

```
        energy_charges = (100 * 5.0) + (200 * 7.0) + ((units_consumed - 300) * 10.0)
    elif type_of_customer_lower == "commercial": if units_consumed <= 200:
        energy_charges = units_consumed * 8.0
      else:
        energy_charges = (200 * 8.0) + ((units_consumed - 200) * 12.0)
    elif type_of_customer_lower == "industrial":
      energy_charges = units_consumed * 15.0
    else:
      # Handle invalid customer type for energy charges
      print(f"Warning: Invalid Type of Customer '{type_of_customer}'. Energy charges might be incorrect.")
      energy_charges = 0.0 # Assign a default or error value

    return energy_charges

# Function to calculate Fixed Charges based on customer type
def calculate_fixed_charges(type_of_customer):
    fixed_charges = 0.0
    # Convert type_of_customer to lowercase for consistent comparison
    type_of_customer_lower = type_of_customer.strip().lower()

    if type_of_customer_lower == "domestic":
    fixed_charges = 50.0 elif
    type_of_customer_lower == "commercial":
    fixed_charges = 100.0 elif
    type_of_customer_lower == "industrial":
    fixed_charges = 200.0

    else:
```

```
        # Handle invalid customer type for fixed charges
        print(f"Warning: Invalid Type of Customer '{type_of_customer}'. Fixed charges might be incorrect.")
        fixed_charges = 0.0 # Assign a default or error value

    return fixed_charges

# Main program flow using the defined functions print("--
- Consumer Billing System (Modular Design) ---")

# Read Previous Units (PU) pu_input =
input("Enter Previous Units (PU): ")
try:
    previous_units = float(pu_input)
except ValueError:
    print("Invalid input for Previous Units. Please enter a number.")
    exit()

# Read Current Units (CU) cu_input =
input("Enter Current Units (CU): ")
try:
    current_units = float(cu_input)
except ValueError:
    print("Invalid input for Current Units. Please enter a number.")
    exit()

# Read Type of Customer
type_of_customer_input = input("Enter Type of Customer (e.g., Domestic, Commercial, Industrial): ")
```

```
# Calculate units consumed if current_units >=
previous_units: units_consumed = current_units -
previous_units

    # Calculate Energy Charges using the function energy_charges =
    calculate_energy_charges(units_consumed, type_of_customer_input)

    # Calculate Fixed Charges using the function fixed_charges =
    calculate_fixed_charges(type_of_customer_input)

    total_bill = energy_charges + fixed_charges

else:
    print("Current Units cannot be less than Previous Units. Please check your input.")
    units_consumed = 0 energy_charges = 0.0 fixed_charges = 0.0 total_bill = 0.0

# Display the results print(f"\n--- Consumer Details and Bill ---")
print(f"Previous Units: {previous_units}") print(f"Current Units:
{current_units}") print(f"Type of Customer:
{type_of_customer_input.capitalize()}") print(f"Units Consumed:
{units_consumed}") print(f"Energy Charges:
${energy_charges:.2f}") print(f"Fixed Charges:
${fixed_charges:.2f}")

print(f"Total Bill: ${total_bill:.2f}")
```

output:-

```
--- Consumer Billing System (Modular Design) ---
Enter Previous Units (PU): 100
```

Enter Current Units (CU): 150

Enter Type of Customer (e.g., Domestic, Commercial, Industrial): Commercial

--- Consumer Details and Bill ---

Previous Units: 100.0

Current Units: 150.0

Type of Customer: Commercial

Units Consumed: 50.0

Energy Charges: $400.00

Fixed Charges: $100.00

Total Bill: $500.00

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

Task Description

Extend the program to calculate:

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

Use AI prompts like:

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

Expected Output

- Individual charge values printed
- Correct duty calculation
- Well-structured output

• Verified intermediate results

Code:-

```
# Function to calculate Energy Charges based on units consumed and customer type
def calculate_energy_charges(units_consumed, type_of_customer):
    energy_charges = 0.0 type_of_customer_lower =
    type_of_customer.strip().lower()

    if type_of_customer_lower == "domestic": if
        units_consumed <= 100: energy_charges
        = units_consumed * 5.0 elif
        units_consumed <= 300:
            energy_charges = (100 * 5.0) + ((units_consumed - 100) * 7.0)
        else:
            energy_charges = (100 * 5.0) + (200 * 7.0) + ((units_consumed - 300) * 10.0)
    elif type_of_customer_lower == "commercial": if units_consumed <= 200:
            energy_charges = units_consumed * 8.0
        else:
            energy_charges = (200 * 8.0) + ((units_consumed - 200) * 12.0)
    elif type_of_customer_lower == "industrial":
        energy_charges = units_consumed * 15.0
    else:
        print(f"Warning: Invalid Type of Customer '{type_of_customer}'. Energy charges might
be incorrect.")
        energy_charges = 0.0
    return energy_charges

# Function to calculate Fixed Charges based on customer type
def calculate_fixed_charges(type_of_customer):
```

```
    fixed_charges = 0.0 type_of_customer_lower =
    type_of_customer.strip().lower()

    if type_of_customer_lower == "domestic":
    fixed_charges = 50.0 elif
    type_of_customer_lower == "commercial":
    fixed_charges = 100.0 elif
    type_of_customer_lower == "industrial":
        fixed_charges = 200.0
    else:
        print(f"Warning: Invalid Type of Customer '{type_of_customer}'. Fixed charges might be
incorrect.")
        fixed_charges = 0.0
    return fixed_charges

# Function to calculate Customer Charges based on customer type
def calculate_customer_charges(type_of_customer):
    customer_charges = 0.0 type_of_customer_lower =
    type_of_customer.strip().lower()

    if type_of_customer_lower == "domestic":
    customer_charges = 20.0 elif
    type_of_customer_lower == "commercial":
    customer_charges = 40.0 elif
    type_of_customer_lower == "industrial":
    customer_charges = 80.0

    else:
        print(f"Warning: Invalid Type of Customer '{type_of_customer}'. Customer charges
```

```
might be incorrect.")
    customer_charges = 0.0 return
    customer_charges

# Main program flow with all charges print("--- Consumer Billing System
(Comprehensive Modular Design) ---")

# Read Previous Units (PU) pu_input =
input("Enter Previous Units (PU): ")
try:
    previous_units = float(pu_input)
except ValueError:
    print("Invalid input for Previous Units. Please enter a number.")
    exit()

# Read Current Units (CU) cu_input =
input("Enter Current Units (CU): ")
try:
    current_units = float(cu_input)
except ValueError:
    print("Invalid input for Current Units. Please enter a number.")
    exit()

# Read Type of Customer
type_of_customer_input = input("Enter Type of Customer (e.g., Domestic, Commercial,
Industrial): ")
# Initialize all charges
units_consumed = 0
energy_charges = 0.0
```

```
fixed_charges = 0.0
customer_charges = 0.0
electricity_duty = 0.0
total_bill = 0.0

# Calculate units consumed and other charges
if current_units >= previous_units:
    units_consumed = current_units - previous_units

    # Calculate charges using functions energy_charges =
    calculate_energy_charges(units_consumed, type_of_customer_input) fixed_charges =
    calculate_fixed_charges(type_of_customer_input) customer_charges =
    calculate_customer_charges(type_of_customer_input)

    # Calculate Electricity Duty (e.g., 5% of Energy Charges)
    ELECTRICITY_DUTY_RATE = 0.05 # 5% electricity_duty =
    energy_charges * ELECTRICITY_DUTY_RATE

    # Calculate Total Bill total_bill = energy_charges + fixed_charges +
    customer_charges + electricity_duty

else: print("Current Units cannot be less than Previous Units. Please check your
    input.")

# Display the results print(f"\n--- Consumer
Details and Total Bill ---") print(f"Previous Units:
{previous_units}") print(f"Current Units:
{current_units}") print(f"Type of Customer:
```

{type_of_customer_input.capitalize()}")

print(f"Units Consumed: {units_consumed}")

print(f"Energy Charges (EC):

${energy_charges:.2f}") print(f"Fixed Charges

(FC): ${fixed_charges:.2f}") print(f"Customer

Charges (CC): ${customer_charges:.2f}")

print(f"Electricity Duty (ED) ({ELECTRICITY_DUTY_RATE*100:.0f}% of EC): ${electricity_duty:.2f}")

print(f"Total Bill: ${total_bill:.2f}")

Output:-

--- Consumer Billing System (Comprehensive Modular Design) ---

Enter Previous Units (PU): 100

Enter Current Units (CU): 150

Enter Type of Customer (e.g., Domestic, Commercial, Industrial): Commercial

--- Consumer Details and Total Bill ---

Previous Units: 100.0

Current Units: 150.0

Type of Customer: Commercial

Units Consumed: 50.0

Energy Charges (EC): $400.00

Fixed Charges (FC): $100.00

Customer Charges (CC): $40.00

Electricity Duty (ED) (5% of EC): $20.00

Total Bill: $560.00

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

• Analyze the program based on:

o Accuracy o Readability

o Real-world applicability

Expected Output

• Complete electricity bill output

• Neatly formatted display

• Sample input/output

• Short analysis paragraph

Program Analysis (Task 5)

Accuracy

The program demonstrates good accuracy in calculating the different components of the electricity bill. The formulas for Energy Charges (EC) are implemented with appropriate slabs for Domestic, Commercial, and Industrial customers. Fixed Charges (FC) and Customer Charges (CC) are applied correctly based on the customer type. Electricity Duty (ED) is calculated as a fixed percentage of the Energy Charges, which is a common real-world practice. Error handling for invalid unit inputs (non-numeric) and the condition where

Current Units are less than Previous Units ensures robust calculations. The use of float for unit inputs allows for precise calculations.

Readability

The program's readability is significantly enhanced due to its modular design, fulfilling the requirements of Task 3 and 4. Breaking down the calculation logic into distinct functions (calculate_energy_charges, calculate_fixed_charges, calculate_customer_charges) makes the code easier to understand, maintain, and debug. Meaningful variable names (previous_units, current_units, units_consumed, type_of_customer_input, energy_charges, fixed_charges, customer_charges, electricity_duty, total_bill) further contribute to clarity. Comments within the functions and the main program flow explain the purpose of different sections. The output formatting is clear and easy to read, presenting each charge individually before summing them up.

Real-world Applicability

This Python application has high real-world applicability for a basic electricity billing system. It covers essential components of a typical electricity bill, including consumption-based charges (EC), fixed costs (FC, CC), and government levies (ED). The ability to differentiate charges based on customer type (Domestic, Commercial, Industrial) is crucial for real-world billing scenarios. While simplified, the structure provides a solid foundation that could be extended to include more complex billing aspects such as:

- **Tax calculations:** e.g., Goods and Services Tax (GST) on various components.
- **Demand charges:** For commercial and industrial customers.
- **Meter reading variations:** Handling estimated readings or faulty meters.
- **Time-of-day (TOD) tariffs:** Different rates for peak and off-peak hours.
- **Database integration:** Storing customer data and billing history.
- **User authentication and management.**

Overall, the program provides a functional and well-structured solution for a simplified electricity billing system, demonstrating how AI assistance can be used to build modular and readable code for practical applications.

After running the above code cell, you will be prompted to enter the values in the output area. Enter your sample inputs there. You can then take a screenshot of the code and its output for your documentation.

After running the above code cell, you will be prompted to enter the values in the output area. Enter your sample inputs there. You can then take a screenshot of the code and its output for your documentation.

Please run the first code cell to define units_consumed and type_of_customer variables before running this new cell. This cell will then calculate and display only the energy charges.