SR UNIVERSITY

AI Assistant Coding
Assignment-3

Name of Student	**: V. Abhinav**	**Batch : 41**
Enrollment No.	**: 2303A52486**	

Task 1: AI-Generated Logic for Reading Consumer Details.

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

- **Reads:**
  - **Previous Units (PU)**
  - **Current Units (CU)**
  - **Type of Customer**
- **Calculates units consumed**
- **Implements logic directly in the main program (no functions)**

Prompt-1:

Generate a Python program (without using functions) to read:
- Previous Units (PU)
- Current Units (CU)
- Type of customer (Domestic / Commercial / Industrial)

Validate inputs and calculate units consumed as:
units_consumed = CU - PU

Print all inputs and the calculated units consumed with clear labels.
Add comments explaining each step.

Prompt-2:

Improve the Python code by adding validation to ensure:
- Current units are greater than previous units
- Customer type input is case-insensitive

Keep all logic inside the main program and avoid functions.

SR UNIVERSITY

Code:

```python
# This script reads Previous Units (PU), Current Units (CU), and customer type,
# validates inputs, computes units_consumed = CU - PU, and prints values.
# No functions are used — the script is a straight-line program with loops for validation.

# Read and validate Previous Units (PU)
while True:
    try:
        pu_input = input("Enter Previous Units (PU): ").strip()  # prompt user
        PU = int(pu_input)  # convert to integer
        if PU < 0:  # ensure non-negative
            print("Previous Units must be zero or positive. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input for PU. Please enter a whole number (e.g., 120).")

# Read and validate Current Units (CU)
while True:
    try:
        cu_input = input("Enter Current Units (CU): ").strip()  # prompt user
        CU = int(cu_input)  # convert to integer
        if CU < 0:  # ensure non-negative
            print("Current Units must be zero or positive. Please try again.")
            continue
        if CU < PU:  # ensure CU >= PU
            print("Current Units cannot be less than Previous Units. Please re-enter.")
            continue
        break
    except ValueError:
        print("Invalid input for CU. Please enter a whole number (e.g., 250).")

# Read and validate Type of customer
# Accepts full names or single-letter abbreviations, case-insensitive.
valid_map = {
    "domestic": "Domestic",
    "d": "Domestic",
    "commercial": "Commercial",
    "c": "Commercial",
    "industrial": "Industrial",
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe "c:/Users/Ab
Lab3.py"
Enter Previous Units (PU): 120
Enter Current Units (CU): 250
Enter Type of customer (Domestic / Commercial / Industrial): Domestic

--- Meter Reading Summary ---
Previous Units (PU): 120
Current Units  (CU): 250
Units Consumed      : 130
Customer Type       : Domestic
PS C:\Users\Abhi\Documents\AI Assistant Coding>
```

Explanation:

This task focuses on building a basic Python program (without functions) to read previous units, current units, and customer type. The AI-generated code correctly validates numeric inputs using loops and exception handling, ensuring current units are greater than previous units. Case-insensitive handling of customer type improves usability and robustness. The output clearly displays all inputs along with the calculated units consumed, demonstrating correct logic implementation.

Task 2: Energy Charges Calculation Based on Units Consumed

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”**

Prompt-1:
Extend the existing Python program to calculate Energy Charges (EC) using conditional statements:

Domestic:
- First 100 units: ₹1.5/unit
- Above 100 units: ₹2.5/unit

Commercial:
- Flat rate ₹4.0/unit

Industrial:
- Flat rate ₹6.0/unit

Use if-elif-else statements and print the calculated EC.

Add meaningful comments.
Give few input examples for these

Prompt-2:
Simplify and optimize the energy charge calculation logic to improve readability.Ensure the conditional structure is clean and easy for students to understand.

Code:

```
# Task-2
# Reads Previous Units (PU), Current Units (CU), and customer type,
# validates inputs, computes units_consumed = CU - PU, calculates Energy Charges (EC),
# and prints all values. This is a straight-line script without functions.

# Read and validate Previous Units (PU)
while True:
    try:
        pu_input = input("Enter Previous Units (PU): ").strip()
        PU = int(pu_input)
        if PU < 0:
            print("Previous Units must be zero or positive. Please try again.")
            continue
        break
    except ValueError:
        print("Invalid input for PU. Please enter a whole number (e.g., 120).")

# Read and validate Current Units (CU)
while True:
    try:
        cu_input = input("Enter Current Units (CU): ").strip()
        CU = int(cu_input)
        if CU < 0:
            print("Current Units must be zero or positive. Please try again.")
            continue
        if CU < PU:
            print("Current Units cannot be less than Previous Units. Please re-enter.")
            continue
        break
    except ValueError:
        print("Invalid input for CU. Please enter a whole number (e.g., 250).")

# Read and validate Type of customer
valid_map = {
    "domestic": "Domestic",
    "d": "Domestic",
    "commercial": "Commercial",
    "c": "Commercial",
    "industrial": "Industrial",
    "i": "Industrial"
```

```python

while True:
    cust_input = input("Enter Type of customer (Domestic / Commercial / Industrial): ").strip().lower()
    if cust_input in valid_map:
        customer_type = valid_map[cust_input]
        break
    else:
        print("Invalid customer type. Enter Domestic, Commercial, Industrial, or D/C/I.")

# Calculate units consumed
units_consumed = CU - PU  # units_consumed = CU - PU

# Calculate Energy Charges (EC) using a clear if/elif/else structure
# Domestic: first 100 units @ 1.5/unit, remaining @ 2.5/unit
# Commercial: flat 4.0/unit
# Industrial: flat 6.0/unit

if customer_type == "Domestic":
    # Compute domestic slab in two parts for clarity
    first_slab_units = min(units_consumed, 100)              # units up to 100
    remaining_units = max(0, units_consumed - 100)           # units above 100
    EC = first_slab_units * 1.5 + remaining_units * 2.5
elif customer_type == "Commercial":
    EC = units_consumed * 4.0
elif customer_type == "Industrial":
    EC = units_consumed * 6.0
else:
    # Fallback (should not occur because input is validated)
    EC = 0.0

# Print all inputs and the calculated values
print("\n--- Meter Reading Summary ---")
print("Previous Units (PU):", PU)
print("Current Units  (CU):", CU)
print("Units Consumed      :", units_consumed)
print("Customer Type       :", customer_type)
print("Energy Charges (EC) : ₹{:.2f}".format(EC))
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe
Lab3.py"
Enter Previous Units (PU): 0
Enter Current Units (CU): 500
Enter Type of customer (Domestic / Commercial / Industrial): Commercial

--- Meter Reading Summary ---
Previous Units (PU): 0
Current Units  (CU): 500
Units Consumed      : 500
Customer Type       : Commercial
Energy Charges (EC) : ₹2000.00
```

Explanation:

In this task, the program is extended to calculate energy charges using conditional statements based on customer type. The tariff rules for Domestic, Commercial, and Industrial consumers are implemented cleanly using `if-elif-else`, improving readability. The optimized logic separates unit slabs for domestic users while applying flat rates for others. Sample outputs confirm accurate charge calculation and proper branching.

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

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**

Prompt-1:

Rewrite the program using user-defined functions.

Create a function named calculate_energy_charges(units, customer_type)
that returns the energy charges based on tariff rules.
Call the function from the main program.
Add proper comments and sample output printing.

Prompt-2:

Add another user-defined function named calculate_fixed_charges(customer_type)
with the following logic:

Domestic: ₹50
Commercial: ₹100
Industrial: ₹150

Return the fixed charges and display them in the main program.
Include comments explaining function usage.

Code:

```python
def calculate_energy_charges(units, customer_type):
    """
    Calculate and return energy charges (EC) based on tariff rules.
    - Domestic: first 100 units @ ₹1.5/unit, remaining @ ₹2.5/unit
    - Commercial: flat ₹4.0/unit
    - Industrial: flat ₹6.0/unit

    Parameters:
      units (int) : number of units consumed (>= 0)
      customer_type (str) : 'Domestic', 'Commercial', or 'Industrial' (case-insensitive)

    Returns:
      float : calculated energy charges
    """
    ct = customer_type.strip().lower()
    if ct == "domestic":
        first_slab = min(units, 100)
        remaining = max(0, units - 100)
        return first_slab * 1.5 + remaining * 2.5
    elif ct == "commercial":
        return units * 4.0
    elif ct == "industrial":
        return units * 6.0
    else:
        return 0.0  # fallback (shouldn't occur with validated input)


# --- Main program starts here ---
if __name__ == "__main__":
    # Read and validate Previous Units (PU)
    while True:
        try:
            PU = int(input("Enter Previous Units (PU): ").strip())
            if PU < 0:
                print("Previous Units must be zero or positive. Try again.")
                continue
            break
        except ValueError:
            print("Invalid input for PU. Enter a whole number (e.g., 120).")
```

```
            print("Previous Units must be zero or positive. Try again.")
            continue
        break
    except ValueError:
        print("Invalid input for PU. Enter a whole number (e.g., 120).")

# Read and validate Current Units (CU)
while True:
    try:
        CU = int(input("Enter Current Units (CU): ").strip())
        if CU < 0:
            print("Current Units must be zero or positive. Try again.")
            continue
        if CU < PU:
            print("Current Units cannot be less than Previous Units. Re-enter.")
            continue
        break
    except ValueError:
        print("Invalid input for CU. Enter a whole number (e.g., 250).")

# Read and validate Type of customer
valid_map = {
    "domestic": "Domestic", "d": "Domestic",
    "commercial": "Commercial", "c": "Commercial",
    "industrial": "Industrial", "i": "Industrial"
}
while True:
    raw = input("Enter Type of customer (Domestic / Commercial / Industrial): ").strip().lower()
    if raw in valid_map:
        customer_type = valid_map[raw]
        break
    else:
        print("Invalid customer type. Enter Domestic, Commercial, Industrial, or D/C/I.")

# Compute units consumed and charges using the functions
units_consumed = CU - PU
EC = calculate_energy_charges(units_consumed, customer_type)
FC = calculate_fixed_charges(customer_type)

# Print summary including Fixed Charges
print("\n--- Meter Reading Summary ---")
print("Previous Units (PU):", PU)
print("Current Units  (CU):", CU)
print("Units Consumed      :", units_consumed)
print("Customer Type       :", customer_type)
print("Energy Charges (EC) : ₹{:.2f}".format(EC))
print("Fixed Charges (FC)  : ₹{:.2f}".format(float(FC)))
print("Total Payable       : ₹{:.2f}".format(EC + FC))
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS                                Python

PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs/Python/Python314/python.exe
/Users/Abhi/Documents/AI Assistant Coding/Lab3.py"
Enter Previous Units (PU): 100
Enter Current Units (CU): 180
Enter Type of customer (Domestic / Commercial / Industrial): Domestic

--- Meter Reading Summary ---
Previous Units (PU): 100
Current Units  (CU): 180
Units Consumed      : 80
Customer Type       : Domestic
Energy Charges (EC) : ₹120.00
```

```
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS

/Users/Abhi/Documents/AI Assistant Coding/Lab3.py"
Enter Previous Units (PU): 0
Enter Current Units (CU): 500
Enter Type of customer (Domestic / Commercial / Industrial): Commercial

--- Meter Reading Summary ---
Previous Units (PU): 0
Current Units  (CU): 500
Units Consumed      : 500
Customer Type       : Commercial
Energy Charges (EC) : ₹2000.00
Fixed Charges (FC)  : ₹100.00
Total Payable       : ₹2100.00
```

Explanation:

This task introduces modular programming by refactoring the earlier logic into user-defined functions. The `calculate_energy_charges()` and `calculate_fixed_charges()` functions encapsulate tariff logic, improving reusability and maintainability. Input validation remains in the main program, keeping responsibilities well separated. The output verifies that functions return correct values and integrate smoothly into the billing flow.

Task 4: Calculation of Additional Charges Scenario

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:

Extend the function-based electricity billing program to calculate:
- Customer Charges (CC) = ₹30 for all consumers
- Electricity Duty (ED) = 5% of Energy Charges (EC)

Print EC, FC, CC, and ED separately with proper formatting.
Add comments for billing accuracy.
Improve billing accuracy by formatting all monetary values to two decimal places. Ensure calculations are clear and correct.

Code:

```
# --- Main program starts here ---
if __name__ == "__main__":
    # Read and validate Previous Units (PU)
    while True:
        try:
            PU = int(input("Enter Previous Units (PU): ").strip())
            if PU < 0:
                print("Previous Units must be zero or positive. Try again.")
                continue
            break
        except ValueError:
            print("Invalid input for PU. Enter a whole number (e.g., 120).")

    # Read and validate Current Units (CU)
    while True:
        try:
            CU = int(input("Enter Current Units (CU): ").strip())
            if CU < 0:
                print("Current Units must be zero or positive. Try again.")
                continue
            if CU < PU:
                print("Current Units cannot be less than Previous Units. Re-enter.")
                continue
            break
        except ValueError:
            print("Invalid input for CU. Enter a whole number (e.g., 250).")

    # Read and validate Type of customer
    valid_map = {
        "domestic": "Domestic", "d": "Domestic",
        "commercial": "Commercial", "c": "Commercial",
        "industrial": "Industrial", "i": "Industrial"
    }
    while True:
        raw = input("Enter Type of customer (Domestic / Commercial / Industrial): ").strip()
        if raw in valid_map:
            customer_type = valid_map[raw]
            break
        else:
```

Output:

```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppData/Local/Programs
/Users/Abhi/Documents/AI Assistant Coding/Assign 3.py"
Enter Previous Units (PU): 400
Enter Current Units (CU): 300
Current Units cannot be less than Previous Units. Re-enter.
Enter Current Units (CU): 500
Enter Type of customer (Domestic / Commercial / Industrial): Commercial

================================================
 ELECTRICITY BILL - DETAILED BREAKDOWN
================================================
Previous Units (PU)       : 400
Current Units  (CU)       : 500
Units Consumed            : 100
Customer Type             : Commercial
------------------------------------------------
Energy Charges (EC)       : ₹400.00
Fixed Charges (FC)        : ₹100.00
Customer Charges (CC)     : ₹30.00
Electricity Duty (ED @ 5%): ₹20.00
------------------------------------------------
TOTAL PAYABLE AMOUNT      : ₹550.00
================================================
```

Explanation:

Here, the billing system is enhanced to include fixed charges, customer charges, and electricity duty. Electricity duty is correctly calculated as a percentage of energy charges, ensuring realistic billing accuracy. All monetary values are formatted to two decimal places, improving professional presentation. The output clearly lists each charge component, making the bill transparent and easy to verify.

Task 5: Final Bill Generation and Output Analysis

- **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**

Prompt:

Now write python code for the following
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

Code:

Assign 3.py > ...

```
416  def calculate_energy_charges(units, customer_type):
421      - Domestic    : ₹1.5/unit for first 100 units, ₹2.5/unit for remaining
422      - Commercial  : ₹4.0/unit (flat rate)
423      - Industrial  : ₹6.0/unit (flat rate)
424
425      Parameters:
426          units (int): Number of units consumed (non-negative)
427          customer_type (str): 'Domestic', 'Commercial', or 'Industrial'
428
429      Returns:
430          float: Energy charges in rupees (₹)
431
432      Accuracy Note: Uses slab calculation for domestic to ensure correct billing
433      """
434      ct = customer_type.strip().lower()
435
436      if ct == "domestic":
437          # Slab-based calculation for accuracy
438          first_slab = min(units, 100)           # First 100 units
439          remaining = max(0, units - 100)        # Units above 100
440          return first_slab * 1.5 + remaining * 2.5
441      elif ct == "commercial":
442          return units * 4.0
443      elif ct == "industrial":
444          return units * 6.0
445      else:
446          return 0.0  # Fallback (should not occur due to validation)
447
448
449  def calculate_fixed_charges(customer_type):
450      """
451      Calculate fixed charges (FC) based on customer type.
452      These are monthly service charges independent of consumption.
453
454      Fixed Charge Structure:
455      - Domestic    : ₹50
456      - Commercial  : ₹100
457      - Industrial  : ₹150
```

Assign 3.py > ...

```python
def get_customer_type():
    valid_map = {
        "domestic": "Domestic", "d": "Domestic",
        "commercial": "Commercial", "c": "Commercial",
        "industrial": "Industrial", "i": "Industrial"
    }

    while True:
        user_input = input("Enter Customer Type (Domestic/Commercial/Industrial or D/C/I
        if user_input in valid_map:
            return valid_map[user_input]
        else:
            print("Invalid type. Enter Domestic, Commercial, Industrial, or D/C/I.")


# Main Program
if __name__ == "__main__":
    print("="*60)
    print("        WELCOME TO ELECTRICITY BILLING SYSTEM")
    print("="*60)

    # Input validation
    PU = get_validated_integer("Enter Previous Units (PU): ", min_value=0)
    CU = get_validated_integer("Enter Current Units (CU): ", min_value=PU,
                               compare_value=PU, compare_type='gte')
    customer_type = get_customer_type()

    # Calculate all billing components
    units_consumed = CU - PU
    EC = calculate_energy_charges(units_consumed, customer_type)
    FC = calculate_fixed_charges(customer_type)
    CC = calculate_customer_charges()
    ED = calculate_electricity_duty(EC)  # 5% of EC only

    # Calculate total: EC + FC + CC + ED
    total_bill = EC + FC + CC + ED

    # Display detailed bill
    display bill(PU, CU, units consumed, customer type, EC, FC, CC, ED, total bill)
```

Output:

```
PS C:\Users\Abhi\Documents\AI Assistant Coding> & C:/Users/Abhi/AppDat
/Users/Abhi/Documents/AI Assistant Coding/Assign 3.py"
============================================================
        WELCOME TO ELECTRICITY BILLING SYSTEM
============================================================
Enter Previous Units (PU): 200
Enter Current Units (CU): 350
Enter Customer Type (Domestic/Commercial/Industrial or D/C/I): Domesti

============================================================
           ELECTRICITY BILL - DETAILED BREAKDOWN
============================================================

METER READING INFORMATION:
------------------------------------------------------------
  Previous Units (PU)        :        200 units
  Current Units  (CU)        :        350 units
  Units Consumed             :        150 units
  Customer Type              :   Domestic

============================================================
BILLING DETAILS:
============================================================
  Energy Charges (EC)        : ₹    275.00
  Fixed Charges (FC)         : ₹     50.00
  Customer Charges (CC)      : ₹     30.00
  Electricity Duty (ED @ 5%) : ₹     13.75
------------------------------------------------------------
  TOTAL PAYABLE AMOUNT       : ₹    368.75
============================================================

Thank you for using our service!
============================================================
```

Explanation:

The final task integrates all components to compute the total payable bill using the formula EC + FC + CC + ED. The program produces a detailed bill breakdown similar to real-world electricity invoices. Accuracy is ensured through validated inputs, structured functions, and precise formatting. Overall, the solution demonstrates strong readability, correctness, and practical applicability.