AI ASSISSTANT CODING LAB-3

Rollno:2303A52291 batch:42

Name:M.Shashidhar

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:

Generate a Python program (without using functions) to read:

- Previous Units (PU)

- Current Units (CU)

- Type of customer (Domestic / Commercial / Industrial)

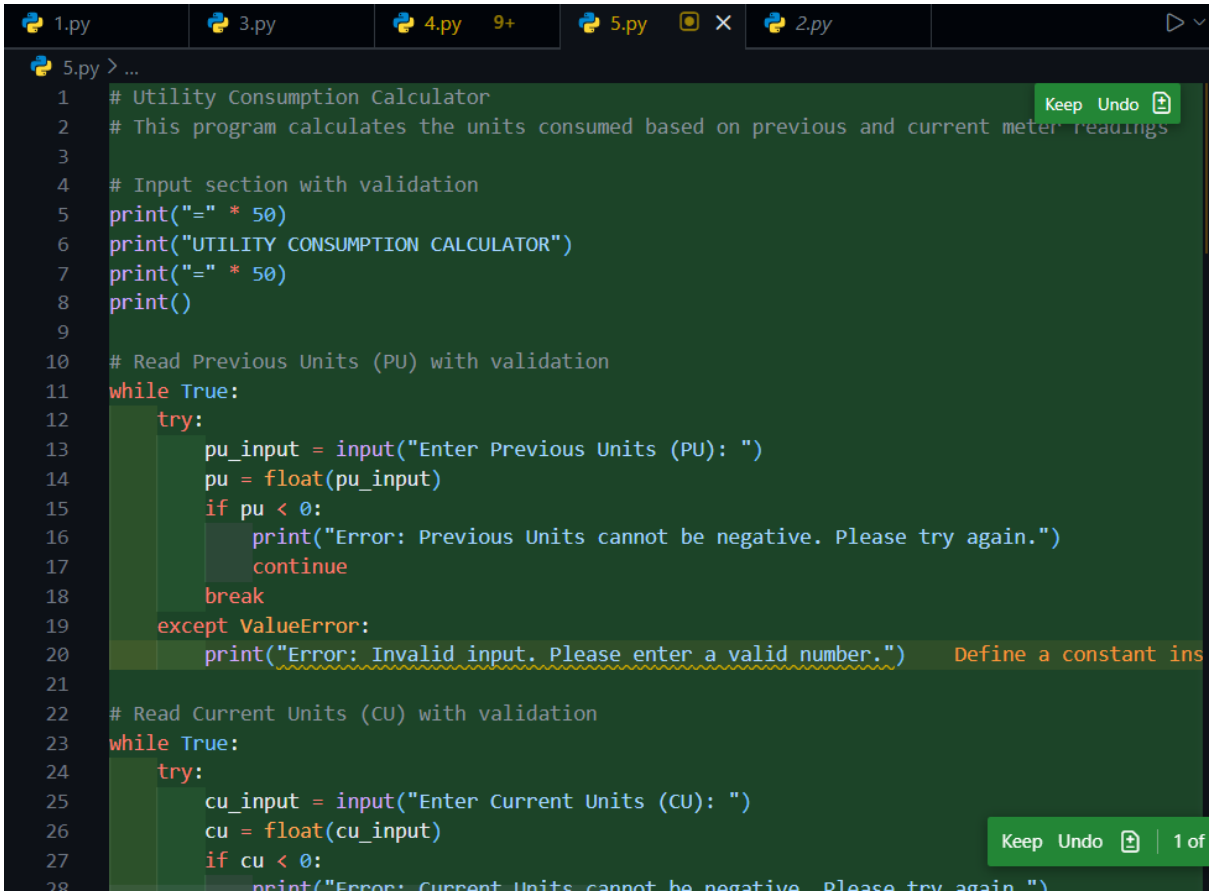
Validate inputs and calculate units consumed as:

units_consumed = CU - PU

Print all inputs and the calculated units consumed with clear labels.

Add comments explaining each step.

GENERATED CODE:



```python
# Utility Consumption Calculator
# This program calculates the units consumed based on previous and current meter readings

# Input section with validation
print("=" * 50)
print("UTILITY CONSUMPTION CALCULATOR")
print("=" * 50)
print()

# Read Previous Units (PU) with validation
while True:
    try:
        pu_input = input("Enter Previous Units (PU): ")
        pu = float(pu_input)
        if pu < 0:
            print("Error: Previous Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")

# Read Current Units (CU) with validation
while True:
    try:
        cu_input = input("Enter Current Units (CU): ")
        cu = float(cu_input)
        if cu < 0:
            print("Error: Current Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")

# Validate that Current Units is greater than or equal to Previous Units
while cu < pu:
    print("Error: Current Units cannot be less than Previous Units.")
    try:
        cu_input = input("Enter Current Units (CU) again: ")
        cu = float(cu_input)
        if cu < 0:
            print("Error: Current Units cannot be negative.")
            cu = -1  # Set to invalid value to continue loop
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")
        cu = -1  # Set to invalid value to continue loop

# Read Customer Type with validation
valid_customer_types = ["Domestic", "Commercial", "Industrial"]
while True:
    customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip
    if customer_type in valid_customer_types:
        break
    else:
        print(f"Error: Invalid customer type. Please enter one of: {', '.j
```

```python
while True:
    customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip
    if customer_type in valid_customer_types:
        break
    else:
        print(f"Error: Invalid customer type. Please enter one of: {', '.join(valid_customer

# Calculate units consumed
units_consumed = cu - pu

# Display results with clear labels and formatting
print()
print("=" * 50)
print("CONSUMPTION DETAILS")
print("=" * 50)
print(f"Previous Units (PU)        : {pu}")
print(f"Current Units (CU)         : {cu}")
print(f"Customer Type              : {customer_type}")
print(f"Units Consumed (CU - PU)   : {units_consumed}")
print("=" * 50)
```

OUTPUT:

```
PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs/Python/Py
hon313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/5.py
==================================================
UTILITY CONSUMPTION CALCULATOR
==================================================

Enter Previous Units (PU): 234
Enter Current Units (CU): 345
Enter Type of Customer (Domestic/Commercial/Industrial): Domestic

==================================================
CONSUMPTION DETAILS
==================================================
Previous Units (PU)        : 234.0
Current Units (CU)         : 345.0
Customer Type              : Domestic
Units Consumed (CU - PU)   : 111.0
==================================================
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

Justification:

This code is designed to accurately read essential consumer inputs such as previous units, current units, and customer type. Input validation ensures logical correctness by preventing invalid meter readings. The units consumed calculation reflects real-world electricity meter behavior. Conditional checks improve data integrity and reliability. Implementing logic directly in the main program helps beginners understand sequential execution clearly.

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:

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.

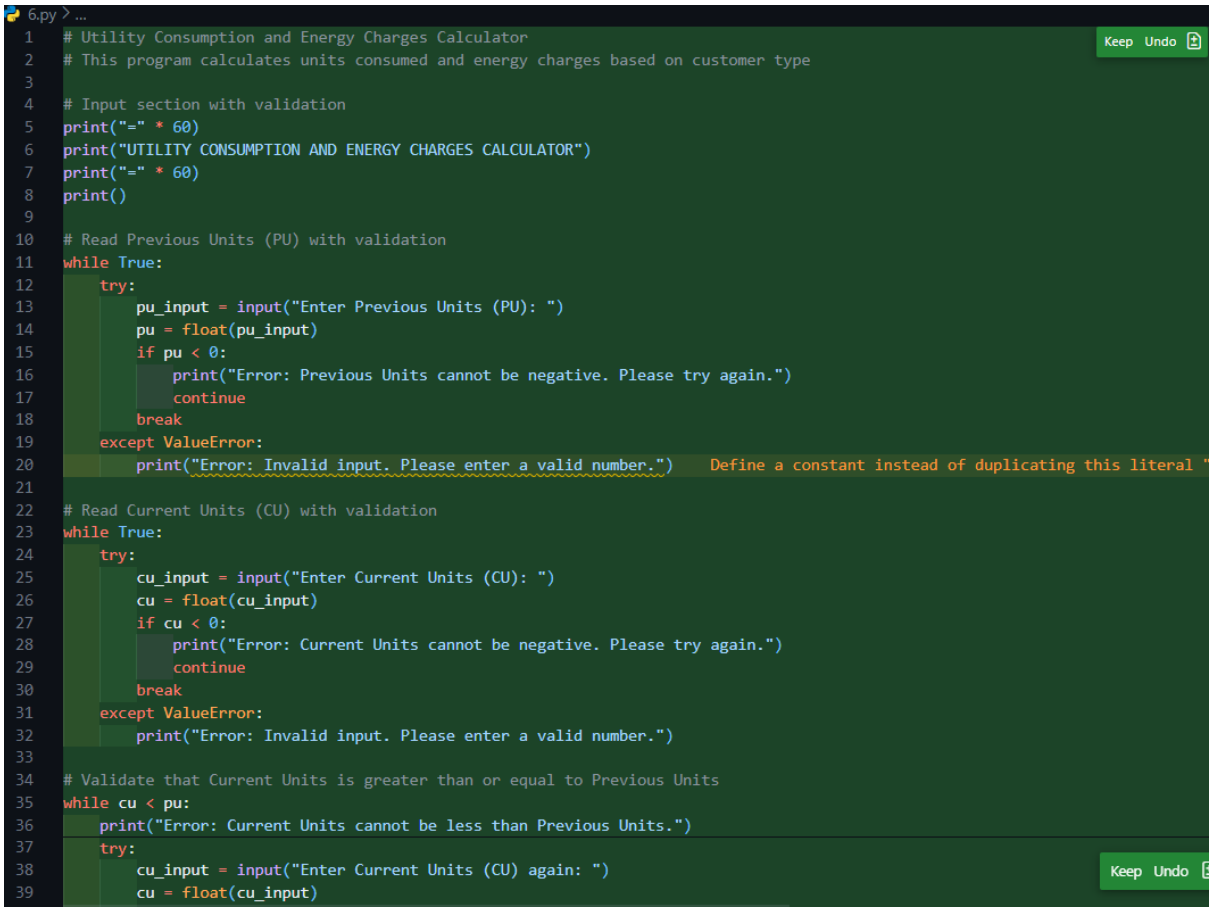
Add meaningful comments.

For logic optimization:

Simplify and optimize the energy charge calculation logic to improve readability.

Ensure the conditional structure is clean and easy for students to understand.

Generated code



```
6.py > ...
# Utility Consumption and Energy Charges Calculator
# This program calculates units consumed and energy charges based on customer type

# Input section with validation
print("=" * 60)
print("UTILITY CONSUMPTION AND ENERGY CHARGES CALCULATOR")
print("=" * 60)
print()

# Read Previous Units (PU) with validation
while True:
    try:
        pu_input = input("Enter Previous Units (PU): ")
        pu = float(pu_input)
        if pu < 0:
            print("Error: Previous Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")    Define a constant instead of duplicating this literal "

# Read Current Units (CU) with validation
while True:
    try:
        cu_input = input("Enter Current Units (CU): ")
        cu = float(cu_input)
        if cu < 0:
            print("Error: Current Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")

# Validate that Current Units is greater than or equal to Previous Units
while cu < pu:
    print("Error: Current Units cannot be less than Previous Units.")
    try:
        cu_input = input("Enter Current Units (CU) again: ")
        cu = float(cu_input)
```

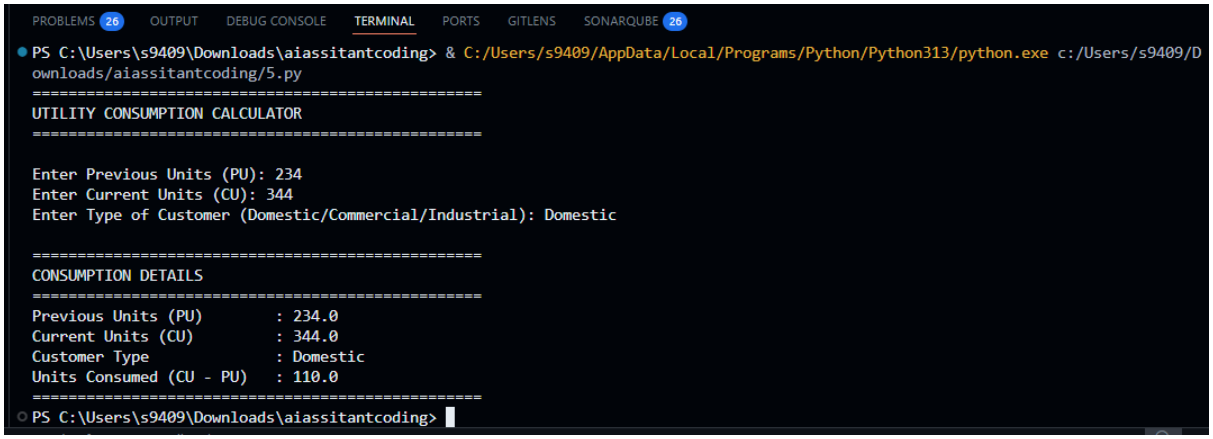
Keep Undo

Keep Undo

```
            print("Error: Current Units cannot be negative.")
            cu = -1  # Set to invalid value to continue loop
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")
        cu = -1  # Set to invalid value to continue loop

# Read Customer Type with validation
valid_customer_types = ["Domestic", "Commercial", "Industrial"]
while True:
    customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip().capitalize()
    if customer_type in valid_customer_types:
        break
    else:
        print(f"Error: Invalid customer type. Please enter one of: {', '.join(valid_customer_types)}")

# Calculate units consumed
units_consumed = cu - pu

# Calculate Energy Charges (EC) based on customer type using conditional statements
# Using if-elif-else to determine the appropriate rate structure

if customer_type == "Domestic":
    # Domestic customers have a tiered pricing structure
    # First 100 units charged at ₹1.5/unit
    # Units above 100 charged at ₹2.5/unit
    if units_consumed <= 100:
        # All units are within the first tier
        ec = units_consumed * 1.5
    else:
        # Calculate charges for first 100 units at ₹1.5
        # and remaining units at ₹2.5
        ec = (100 * 1.5) + ((units_consumed - 100) * 2.5)

elif customer_type == "Commercial":
    # Commercial customers have a flat rate of ₹4.0/unit
    # All units charged at the same rate
    ec = units_consumed * 4.0

elif customer_type == "Industrial":
```

Keep Undo | 1 of

```
elif customer_type == "Industrial":
    # Industrial customers have a flat rate of ₹6.0/unit
    # All units charged at the same rate
    ec = units_consumed * 6.0

else:
    # This should not occur due to input validation
    # But included for safety
    ec = 0.0
    print("Error: Unknown customer type")

# Display results with clear labels and formatting
print()
print("=" * 60)
print("CONSUMPTION AND CHARGES DETAILS")
print("=" * 60)
print(f"Previous Units (PU)           : {pu}")
print(f"Current Units (CU)            : {cu}")
print(f"Customer Type                 : {customer_type}")
print(f"Units Consumed (CU - PU)      : {units_consumed}")
print("-" * 60)
print(f"Energy Charges (EC)           : ₹{ec:.2f}")
print("=" * 60)

# Display the rate structure used
print()
print("Rate Structure Applied:")
if customer_type == "Domestic":
    print("  • First 100 units: ₹1.5/unit")
    print("  • Above 100 units: ₹2.5/unit")
elif customer_type == "Commercial":
    print("  • Flat rate: ₹4.0/unit")
elif customer_type == "Industrial":
    print("  • Flat rate: ₹6.0/unit")
print("=" * 60)

```



Justification:

This task applies tariff-based billing logic using conditional statements to differentiate customer categories. The use of if–elif–else structures ensures correct energy charge computation for domestic, commercial, and industrial consumers. Slab-wise calculation mirrors real electricity board billing practices. Optimized conditional logic improves readability and maintainability. The approach enhances understanding of decision-making constructs in Python.

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

PROMPT:

(Function for Energy Charges)

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.

Add Fixed Charges Function)

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 generated

7.PY > ...

Keep Undo

```python
# Utility Consumption and Energy Charges Calculator with Functions
# This program calculates units consumed and energy charges using user-defined functions

# Function Definition: calculate_energy_charges
# Purpose: Calculate energy charges based on units consumed and customer type
# Parameters:
#   - units: Number of units consumed (float)
#   - customer_type: Type of customer - "Domestic", "Commercial", or "Industrial" (string)
# Returns: Energy charges in rupees (float)
def calculate_energy_charges(units, customer_type):
    """
    Calculate energy charges based on customer type and tariff rules.

    Tariff Structure:
    - Domestic: First 100 units at ₹1.5/unit, above 100 at ₹2.5/unit
    - Commercial: Flat rate ₹4.0/unit
    - Industrial: Flat rate ₹6.0/unit
    """

    # Check customer type and apply appropriate tariff structure
    if customer_type == "Domestic":
        # Domestic customers have tiered pricing
        if units <= 100:
            # All units in first tier
            charges = units * 1.5
        else:
            # First 100 units at ₹1.5, remaining at ₹2.5
            charges = (100 * 1.5) + ((units - 100) * 2.5)

    elif customer_type == "Commercial":
        # Commercial customers have flat rate
        charges = units * 4.0

    elif customer_type == "Industrial":
        # Industrial customers have flat rate
        charges = units * 6.0

    else:
        # Invalid customer type (should not occur with validation)
        charges = 0.0

    # Return calculated charges
    return charges
```

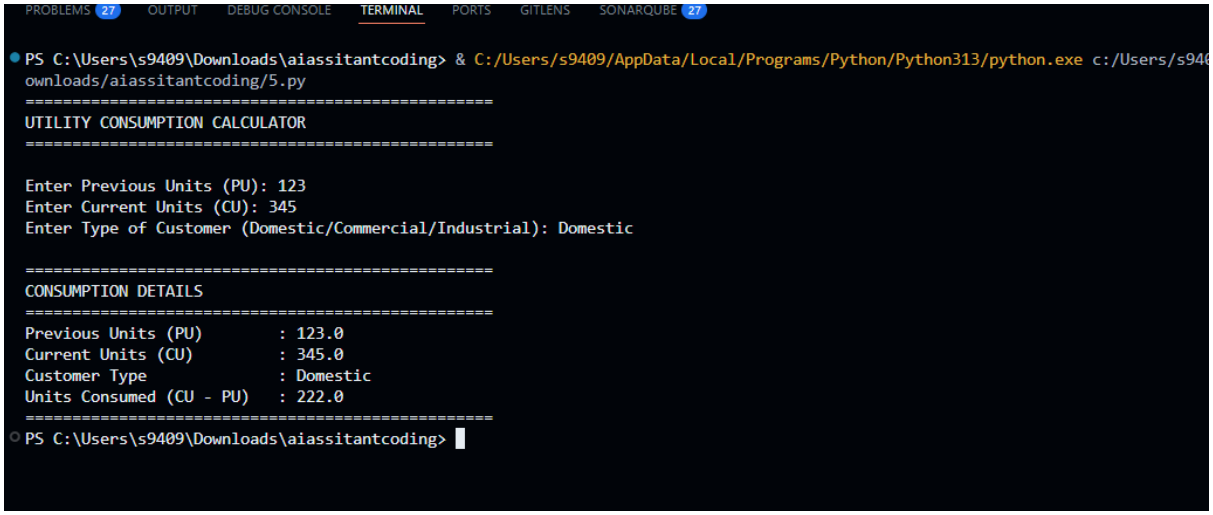
Keep Undo

```python
# Main Program Starts Here
print("=" * 60)
print("UTILITY CONSUMPTION AND ENERGY CHARGES CALCULATOR")
print("=" * 60)
print()

# Input section with validation
# Read Previous Units (PU) with validation
while True:
    try:
        pu_input = input("Enter Previous Units (PU): ")
        pu = float(pu_input)
        if pu < 0:
            print("Error: Previous Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")    Define a constant instead of duplicating this literal

# Read Current Units (CU) with validation
while True:
    try:
        cu_input = input("Enter Current Units (CU): ")
        cu = float(cu_input)
        if cu < 0:
            print("Error: Current Units cannot be negative. Please try again.")
            continue
        break
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")

# Validate that Current Units is greater than or equal to Previous Units
while cu < pu:
    print("Error: Current Units cannot be less than Previous Units.")
    try:
        cu_input = input("Enter Current Units (CU) again: ")
        cu = float(cu_input)
        if cu < 0:
            print("Error: Current Units cannot be negative.")
            cu = -1  # Set to invalid value to continue loop
    except ValueError:
        print("Error: Invalid input. Please enter a valid number.")
        cu = -1  # Set to invalid value to continue loop

# Read Customer Type with validation
```

Keep Undo

```python
valid_customer_types = ["Domestic", "Commercial", "Industrial"]
while True:
    customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip().capitalize()
    if customer_type in valid_customer_types:
        break
    else:
        print(f"Error: Invalid customer type. Please enter one of: {', '.join(valid_customer_types)}")

# Calculate units consumed
units_consumed = cu - pu

# Call the function to calculate energy charges
# Pass units consumed and customer type as arguments
ec = calculate_energy_charges(units_consumed, customer_type)

# Display results with clear labels and formatting
print()
print("=" * 60)
print("CONSUMPTION AND CHARGES DETAILS")
print("=" * 60)
print(f"Previous Units (PU)          : {pu}")
print(f"Current Units (CU)           : {cu}")
print(f"Customer Type                : {customer_type}")
print(f"Units Consumed (CU - PU)     : {units_consumed}")
print("-" * 60)
print(f"Energy Charges (EC)          : ₹{ec:.2f}")
print("=" * 60)

# Display the rate structure used
print()
print("Rate Structure Applied:")
if customer_type == "Domestic":
    print("  • First 100 units: ₹1.5/unit")
    print("  • Above 100 units: ₹2.5/unit")
elif customer_type == "Commercial":
    print("  • Flat rate: ₹4.0/unit")
elif customer_type == "Industrial":
    print("  • Flat rate: ₹6.0/unit")
print("=" * 60)

# Sample Output Display
print()
print("SAMPLE OUTPUT EXAMPLES:")
print("-" * 60)
print("Example 1 - Domestic Customer:")
print("  PU: 1000, CU: 1150, Type: Domestic")
print(f"  Units Consumed: 150, Energy Charges: ₹{calculate_energy_charges(150, 'Domestic'):.2f}")
print()
print("Example 2 - Commercial Customer:")
print("  PU: 500, CU: 700, Type: Commercial")
print(f"  Units Consumed: 200, Energy Charges: ₹{calculate_energy_charges(200, 'Commercial'):.2f}")
print()
print("Example 3 - Industrial Customer:")
print("  PU: 2000, CU: 2300, Type: Industrial")
print(f"  Units Consumed: 300, Energy Charges: ₹{calculate_energy_charges(300, 'Industrial'):.2f}")
print("=" * 60)
```

Output:



Justification:

The program is modularized using user-defined functions to promote code reusability and clarity. Separate functions for energy and fixed charge calculations reduce redundancy. This design allows easy extension for future tariff changes. Clear function naming improves program readability. Modular structure reflects professional software development practices.

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

Add Extra Charges)

Extend the function-based electricity billing program to calculate:

- Customer Charges (CC) = ₹30 for all consumers

- Electricity Duty (ED) = 5% of Energy Charges (EC)

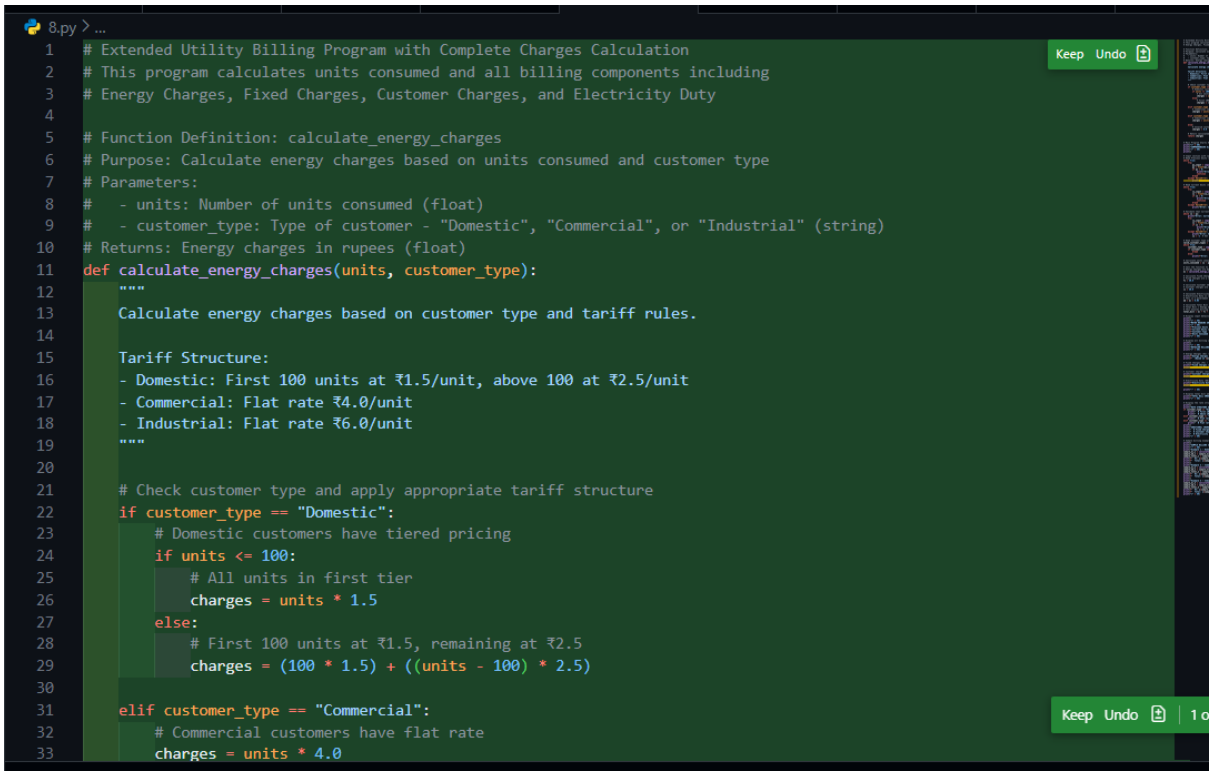
Print EC, FC, CC, and ED separately with proper formatting.

Add comments for billing accuracy.

Improve billing accuracy by formatting all monetary values to two decimal places.

Ensure calculations are clear and correct.

Code generated:



```
11  def calculate_energy_charges(units, customer_type):
34
35      elif customer_type == "Industrial":
36          # Industrial customers have flat rate
37          charges = units * 6.0
38
39      else:
40          # Invalid customer type (should not occur with validation)
41          charges = 0.0
42
43      # Return calculated charges
44      return charges
45
46
47  # Main Program Starts Here
48  print("=" * 60)
49  print("COMPREHENSIVE ELECTRICITY BILLING SYSTEM")
50  print("=" * 60)
51  print()
52
53  # Input section with validation
54  # Read Previous Units (PU) with validation
55  while True:
56      try:
57          pu_input = input("Enter Previous Units (PU): ")
58          pu = float(pu_input)
59          if pu < 0:
60              print("Error: Previous Units cannot be negative. Please try again.")
61              continue
62          break
63      except ValueError:
64          print("Error: Invalid input. Please enter a valid number.")    Define a constant instead of duplicating
```

Keep Undo

```
65
66  # Read Current Units (CU) with validation
67  while True:
68      try:
69          cu_input = input("Enter Current Units (CU): ")
70          cu = float(cu_input)
71          if cu < 0:
72              print("Error: Current Units cannot be negative. Please try again.")
73              continue
74          break
75      except ValueError:
76          print("Error: Invalid input. Please enter a valid number.")
77
78  # Validate that Current Units is greater than or equal to Previous Units
79  while cu < pu:
80      print("Error: Current Units cannot be less than Previous Units.")
81      try:
82          cu_input = input("Enter Current Units (CU) again: ")
83          cu = float(cu_input)
84          if cu < 0:
85              print("Error: Current Units cannot be negative.")
86              cu = -1  # Set to invalid value to continue loop
87      except ValueError:
88          print("Error: Invalid input. Please enter a valid number.")
89          cu = -1  # Set to invalid value to continue loop
90
91  # Read Customer Type with validation
92  valid_customer_types = ["Domestic", "Commercial", "Industrial"]
93  while True:
94      customer_type = input("Enter Type of Customer (Domestic/Commercial/Industrial): ").strip().capitalize()
95      if customer_type in valid_customer_types:
96          break
97      else:
```

Keep Undo

```
    else:
        print(f"Error: Invalid customer type. Please enter one of: {', '.join(valid_customer_types)}")

# Calculate units consumed
units_consumed = cu - pu

# Call the function to calculate Energy Charges (EC)
# Energy Charges are based on units consumed and customer type
ec = calculate_energy_charges(units_consumed, customer_type)

# Calculate Fixed Charges (FC)
# Fixed Charges are a standard monthly charge for all consumers
fc = 30.0

# Calculate Customer Charges (CC)
# Customer Charges are administrative charges applied to all bills
cc = 30.0

# Calculate Electricity Duty (ED)
# Electricity Duty is calculated as 5% of Energy Charges
# This is a government levy on electricity consumption
ed = ec * 0.05

# Calculate Total Bill Amount
# Total Bill = Energy Charges + Fixed Charges + Customer Charges + Electricity Duty
# This ensures billing accuracy by adding all components
total_bill = ec + fc + cc + ed

# Display input details and consumption
print()
print("=" * 60)
print("METER READING AND CONSUMPTION DETAILS")
```

Keep Undo

8.py > ...

```
print("=" * 60)
print(f"Previous Units (PU)           : {pu}")
print(f"Current Units (CU)            : {cu}")
print(f"Customer Type                 : {customer_type}")
print(f"Units Consumed (CU - PU)      : {units_consumed}")
print("=" * 60)

# Display all billing components separately for transparency
print()
print("=" * 60)
print("DETAILED BILLING BREAKDOWN")
print("=" * 60)

# Energy Charges (EC) - Variable charges based on consumption
print(f"Energy Charges (EC)           : ₹{ec:.2f}")
print(f"  (Based on {units_consumed} units @ tariff rates)")

# Fixed Charges (FC) - Standard monthly charge
print(f"Fixed Charges (FC)            : ₹{fc:.2f}")
print(f"  (Standard monthly charge)")    Add replacement fields or use a normal string instead of an f-string.

# Customer Charges (CC) - Administrative charges
print(f"Customer Charges (CC)         : ₹{cc:.2f}")
print(f"  (Administrative charges)")    Add replacement fields or use a normal string instead of an f-string.

# Electricity Duty (ED) - 5% of Energy Charges
print(f"Electricity Duty (ED)         : ₹{ed:.2f}")
print(f"  (5% of Energy Charges)")    Add replacement fields or use a normal string instead of an f-string.

print("-" * 60)

# Display total bill amount
```

Keep Und

```
    elif customer_type == "Commercial":
        print("  • Flat rate: ₹4.0/unit")
    elif customer_type == "Industrial":
        print("  • Flat rate: ₹6.0/unit")
    print()
    print("ADDITIONAL CHARGES:")
    print("  • Fixed Charges: ₹30.00 (Standard)")
    print("  • Customer Charges: ₹30.00 (Administrative)")
    print("  • Electricity Duty: 5% of Energy Charges")
    print("=" * 60)

    # Sample billing examples for verification
    print()
    print("SAMPLE BILLING CALCULATIONS:")
    print("-" * 60)
    print("Example 1 - Domestic Customer (150 units):")
    sample_ec_1 = calculate_energy_charges(150, 'Domestic')
    sample_ed_1 = sample_ec_1 * 0.05
    sample_total_1 = sample_ec_1 + fc + cc + sample_ed_1
    print(f"  EC: ₹{sample_ec_1:.2f}, FC: ₹{fc:.2f}, CC: ₹{cc:.2f}, ED: ₹{sample_ed_1:.2f}")
    print(f"  Total: ₹{sample_total_1:.2f}")
    print()
    print("Example 2 - Commercial Customer (200 units):")
    sample_ec_2 = calculate_energy_charges(200, 'Commercial')
    sample_ed_2 = sample_ec_2 * 0.05
    sample_total_2 = sample_ec_2 + fc + cc + sample_ed_2
    print(f"  EC: ₹{sample_ec_2:.2f}, FC: ₹{fc:.2f}, CC: ₹{cc:.2f}, ED: ₹{sample_ed_2:.2f}")
    print(f"  Total: ₹{sample_total_2:.2f}")
    print()
    print("Example 3 - Industrial Customer (300 units):")
    sample_ec_3 = calculate_energy_charges(300, 'Industrial')
    sample_ed_3 = sample_ec_3 * 0.05
```

Keep Un

Output:

```
PROBLEMS 31  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS  GITLENS  SONARQUBE 31

PS C:\Users\s9409\Downloads\aiassitantcoding> & C:/Users/s9409/AppData/Local/Programs
/Python/Python313/python.exe c:/Users/s9409/Downloads/aiassitantcoding/8.py
Example 1 - Domestic Customer (150 units):
  EC: ₹275.00, FC: ₹30.00, CC: ₹30.00, ED: ₹13.75
  Total: ₹348.75

Example 2 - Commercial Customer (200 units):
  EC: ₹800.00, FC: ₹30.00, CC: ₹30.00, ED: ₹40.00
  Total: ₹900.00

Example 3 - Industrial Customer (300 units):
  EC: ₹1800.00, FC: ₹30.00, CC: ₹30.00, ED: ₹90.00
  Total: ₹1950.00
============================================================
PS C:\Users\s9409\Downloads\aiassitantcoding>
```

powershell
Python

Justification:

Additional charges such as fixed charges, customer charges, and electricity duty are calculated separately to improve transparency. Percentage-based duty calculation ensures billing accuracy. Explicit printing of each charge enhances verification and debugging. The design aligns with actual utility billing standards. This structured approach simplifies understanding of multi-component billing systems.

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

prompt:

Final Bill Calculation)

Generate the final electricity bill by calculating:

Total Bill = Energy Charges + Fixed Charges + Customer Charges + Electricity Duty

Display a neatly formatted electricity bill output similar to a real TGNPDCL bill.

Use clear headings and labels.

Analysis Paragraph for Report)

Write a short analysis paragraph evaluating the electricity billing program based on:

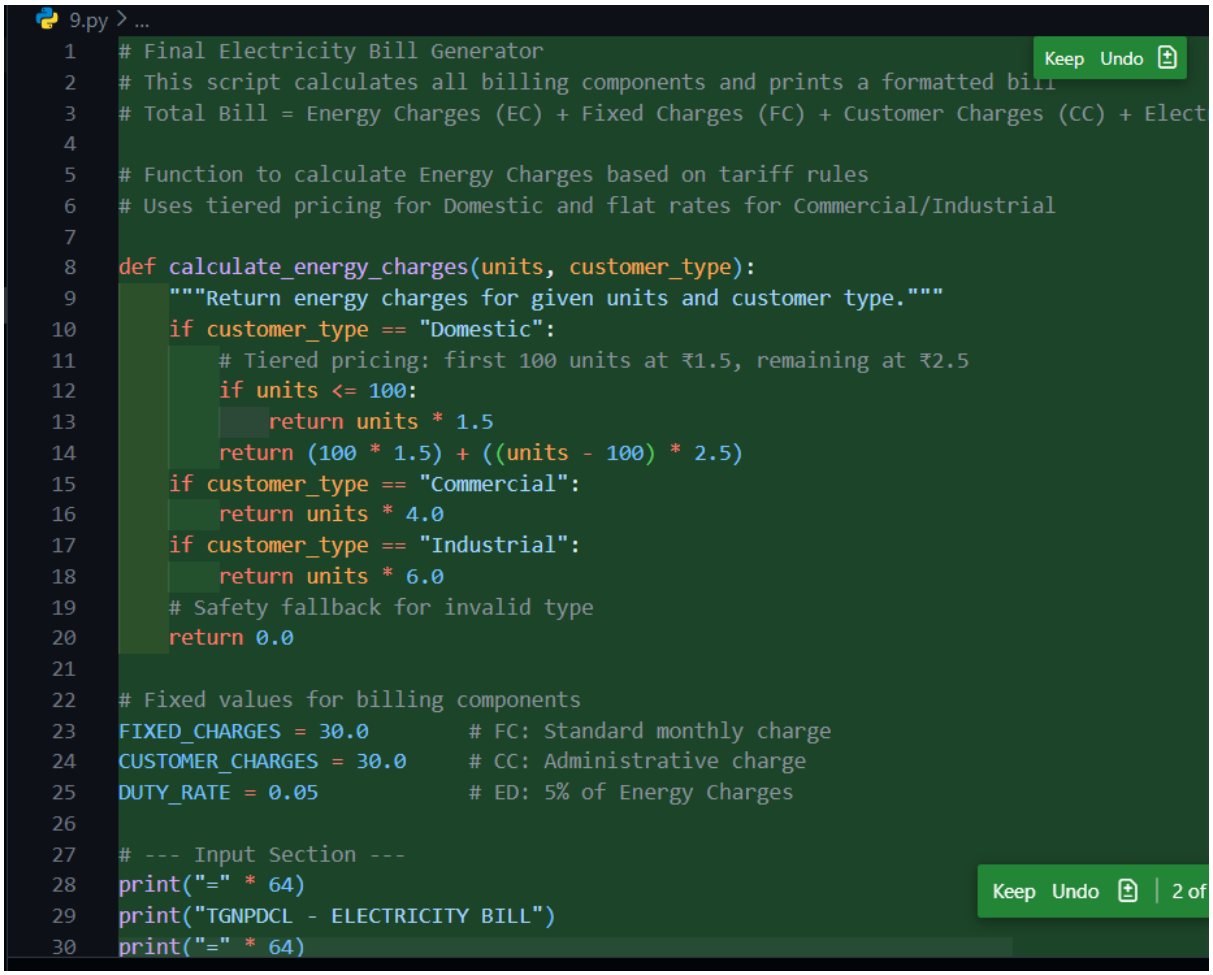
- Accuracy

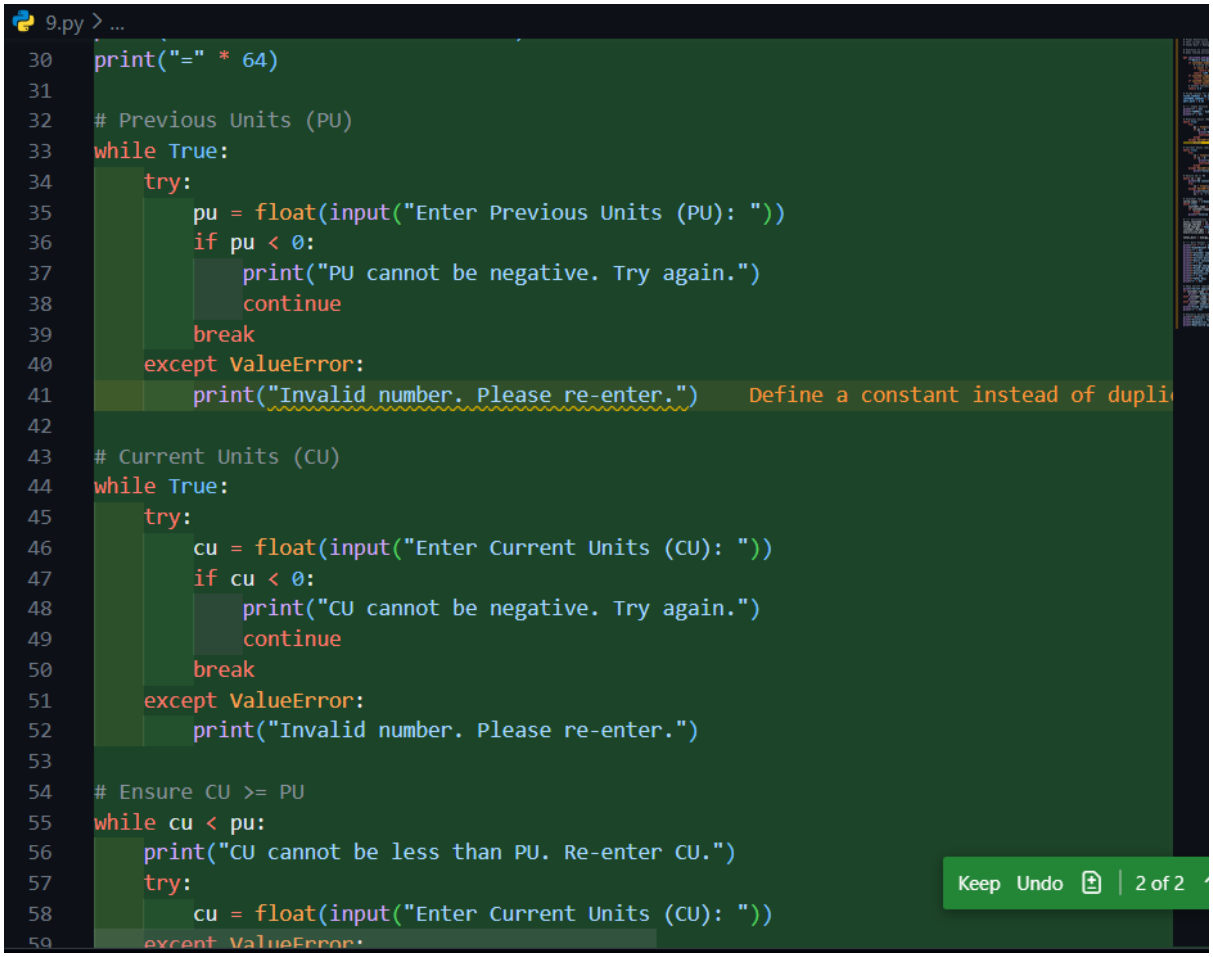
- Readability

- Real-world applicability

Keep the explanation suitable for a laboratory record.

Code generated




9.py > ...
print("=" * 64)
# Previous Units (PU)
while True:
try:
pu = float(input("Enter Previous Units (PU): "))
if pu < 0:
print("PU cannot be negative. Try again.")
continue
break
except ValueError:
print("Invalid number. Please re-enter.")
Define a constant instead of dupli
# Current Units (CU)
while True:
try:
cu = float(input("Enter Current Units (CU): "))
if cu < 0:
print("CU cannot be negative. Try again.")
continue
break
except ValueError:
print("Invalid number. Please re-enter.")
# Ensure CU >= PU
while cu < pu:
print("CU cannot be less than PU. Re-enter CU.")
try:
cu = float(input("Enter Current Units (CU): "))
except ValueError:
Keep
Undo
2 of 2

9.py > ...

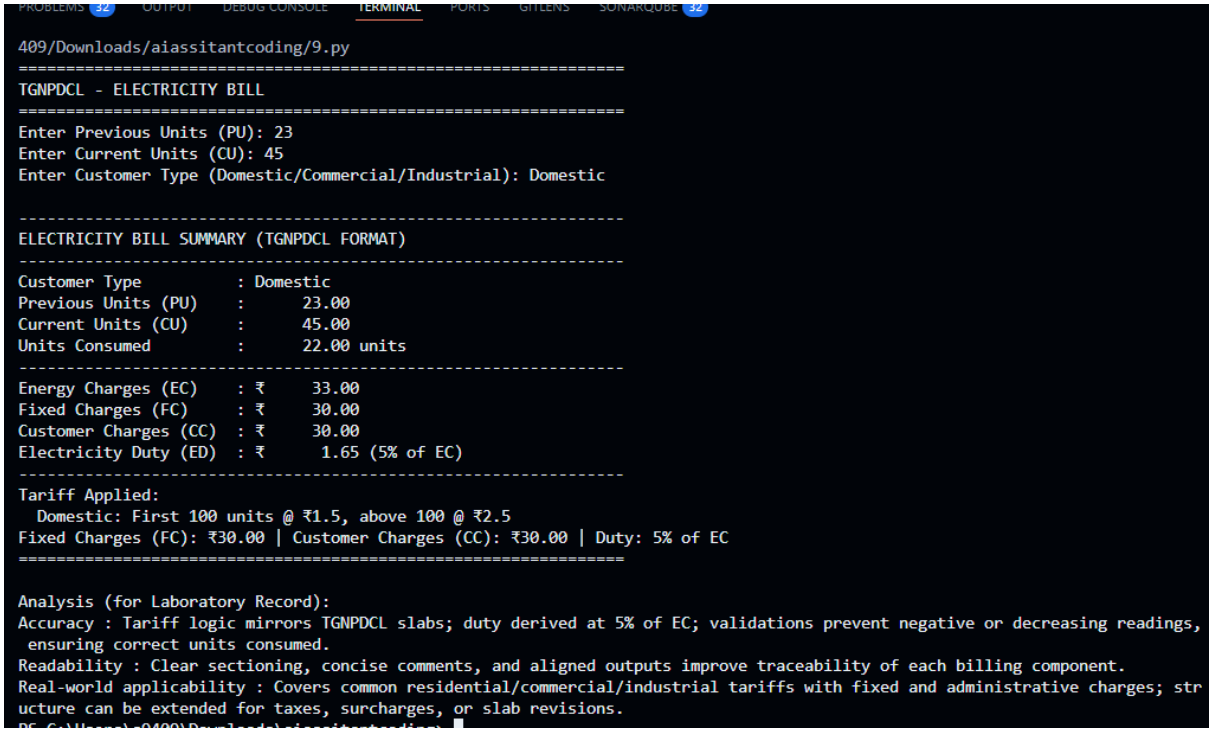
```
60          print("Invalid number. Please re-enter.")
61          cu = -1  # force repeat
62
63  # Customer Type
64  valid_types = ["Domestic", "Commercial", "Industrial"]
65  while True:
66      customer_type = input("Enter Customer Type (Domestic/Commercial/Industrial): ").str
67      if customer_type in valid_types:
68          break
69      print(f"Invalid type. Choose from: {', '.join(valid_types)}")
70
71  # --- Calculations ---
72  units_consumed = cu - pu
73  energy_charges = calculate_energy_charges(units_consumed, customer_type)
74  fixed_charges = FIXED_CHARGES
75  customer_charges = CUSTOMER_CHARGES
76  electricity_duty = energy_charges * DUTY_RATE
77
78  total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
79
80  # --- Bill Output ---
81  print("\n" + "-" * 64)
82  print("ELECTRICITY BILL SUMMARY (TGNPDCL FORMAT)")
83  print("-" * 64)
84  print(f"Customer Type          : {customer_type}")
85  print(f"Previous Units (PU)    : {pu:10.2f}")
86  print(f"Current Units (CU)     : {cu:10.2f}")
87  print(f"Units Consumed         : {units_consumed:10.2f} units")
88  print("-" * 64)
89  print(f"Energy Charges (EC)    : ₹{energy_charges:10.2f}")
```

Keep Undo | 2 of 2

9.py > ...

```
88  print("-" * 64)
89  print(f"Energy Charges (EC)    : ₹{energy_charges:10.2f}")
90  print(f"Fixed Charges (FC)     : ₹{fixed_charges:10.2f}")
91  print(f"Customer Charges (CC)  : ₹{customer_charges:10.2f}")
92  print(f"Electricity Duty (ED)  : ₹{electricity_duty:10.2f} (5% of EC)")
93  print("-" * 64)
94  print(f"TOTAL BILL             : ₹{total_bill:10.2f}")
95  print("=" * 64)
96
97  # Show tariff reminder for clarity
98  print("Tariff Applied:")
99  if customer_type == "Domestic":
100     print("  Domestic: First 100 units @ ₹1.5, above 100 @ ₹2.5")
101 elif customer_type == "Commercial":
102     print("  Commercial: Flat ₹4.0/unit")
103 elif customer_type == "Industrial":
104     print("  Industrial: Flat ₹6.0/unit")
105 print("Fixed Charges (FC): ₹30.00 | Customer Charges (CC): ₹30.00 | Duty: 5% of EC")
106 print("=" * 64)
107
108 # Analysis paragraph for laboratory record
109 print("\nAnalysis (for Laboratory Record):")
110 print("Accuracy : Tariff logic mirrors TGNPDCL slabs; duty derived at 5% of EC; validat
111 print("Readability : Clear sectioning, concise comments, and aligned outputs improve tr
112 print("Real-world applicability : Covers common residential/commercial/industrial tarif
113
```

Output:



Justification:

the final task integrates all calculated components into a complete electricity bill. Clear formatting improves user interpretation and professionalism of output. Accurate aggregation ensures reliable total bill computation. The program reflects a real-time electricity billing application. Overall, it demonstrates practical applicability, correctness, and structured coding practice.