

AI Assisted Coding Lab 3.3

Veekshitha Adharasani

2303A52175

Batch - 41

Question 1:

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

Prompt:

Generate a code for reading consumer details scenario. The electricity billing system must collect accurate consumer data. It should read Previous Units (PU), Current Units (CU), Type of Customer. It should calculate units consumed. It should implement logic directly in the main program with no functions. The output should be correct input reading, units consumed calculation. And the customers should be at least 10. The code should be more efficient and readable. Also, generate the consumer data automatically within the code instead of taking manual input. And print the details for every customer separately.

CODE:

```
import random
# Each record includes Previous Units (PU), Current Units (CU), and Type of Customer
customers_data = []
num_customers = 12 # Ensure at least 10 customers
for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })
# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Unit Consumption\n")
print(f"Total Customers Processed: {len(customers_data)}\n")
for customer in customers_data:
    # Read input values
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']
    customer_type = customer['customer_type']
    # Calculate units consumed
    units_consumed = current_units - previous_units
    # Display the output for each customer separately
    print("-----")
    print(f"Customer ID: {customer_id}")
    print(f"Customer Type: {customer_type}")
    print(f"Previous Units: {previous_units}")
    print(f"Current Units: {current_units}")
    print(f"Units Consumed: {units_consumed}")
    print("-----\n")
```

```
previous_units = random.randint(0, 1000)
current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

customers_data.append({
    'customer_id': f'{i:03d}',
    'previous_units': previous_units,
    'current_units': current_units,
    'customer_type': customer_type
})
# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Unit Consumption\n")
print(f"Total Customers Processed: {len(customers_data)}\n")
for customer in customers_data:
    # Read input values
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']
    customer_type = customer['customer_type']
    # Calculate units consumed
    units_consumed = current_units - previous_units
    # Display the output for each customer separately
    print("-----")
    print(f"Customer ID: {customer_id}")
    print(f"Customer Type: {customer_type}")
    print(f"Previous Units: {previous_units}")
    print(f"Current Units: {current_units}")
    print(f"Units Consumed: {units_consumed}")
    print("-----\n")
```

OUTPUT:

```
Electricity Billing System - Consumer Details and Unit Consumption
...
Total Customers Processed: 12

-----
Customer ID: C001
Customer Type: Commercial
Previous Units: 747
Current Units: 1023
Units Consumed: 276
-----

-----
Customer ID: C002
Customer Type: Commercial
Previous Units: 854
Current Units: 1330
Units Consumed: 476
-----

-----
Customer ID: C003
Customer Type: Commercial
Previous Units: 729
Current Units: 1051
Units Consumed: 322
-----

-----
Customer ID: C004
Customer Type: Residential
...
```

```
Customer ID: C005
Customer Type: Commercial
Previous Units: 629
Current Units: 750
Units Consumed: 121
-----

-----
Customer ID: C006
Customer Type: Industrial
Previous Units: 170
Current Units: 466
Units Consumed: 296
-----

-----
Customer ID: C007
Customer Type: Industrial
Previous Units: 690
Current Units: 819
Units Consumed: 129
-----

-----
Customer ID: C008
Customer Type: Commercial
Previous Units: 905
Current Units: 1126
Units Consumed: 221
-----

-----
Customer ID: C009
Customer Type: Residential
Previous Units: 148
Current Units: 542
Units Consumed: 394
-----

-----
Customer ID: C010
Customer Type: Industrial
Previous Units: 645
Current Units: 812
Units Consumed: 167
-----

-----
Customer ID: C011
Customer Type: Commercial
Previous Units: 264
Current Units: 489
Units Consumed: 225
-----

-----
Customer ID: C012
Customer Type: Industrial
Previous Units: 623
Current Units: 973
Units Consumed: 350
-----
```

JUSTIFICATION:

This task ensures accurate simulation of a real-world electricity billing scenario by systematically collecting consumer data such as previous units, current units, and customer type. Automatically generating data avoids manual input errors and improves testing reliability. Writing logic directly in the main program enhances transparency and makes the execution flow easy to understand for learners. The calculation of units consumed demonstrates correct use of arithmetic and data handling. Processing at least ten customers reflects practical scalability. Structured output improves readability and mimics real billing systems. The program also reinforces clean coding practices.

Question 2:

Task 2: Energy Charges Calculation Based on Units Consumed Scenario

Prompt:

Generate a code for energy charges calculation based on units consumed scenario. The Energy charges depend on the number of units consumed and customer type. Extend the previous code, that it calculates Energy Charges(EC). Use conditional statements based on domestic, commercial, industrial consumers. Simplify the energy charge calculation logic. Optimize conditional statements. Generate a efficient code and improve readability.

CODE:

```
import random
# Each record includes Previous Units (PU), Current Units (CU), and Type of Customer
customers_data = []
num_customers = 12 # Ensure at least 10 customers
for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'C{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })

# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Unit Consumption\n")
print(f"Total Customers Processed: {len(customers_data)}\n")
for customer in customers_data:
    # Read input values
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']
    customer_type = customer['customer_type']

    # Calculate units consumed
    units_consumed = current_units - previous_units

    # Calculate Energy Charges (EC) based on customer type and units consumed
    energy_charges = 0.0
    if customer_type == 'Residential':
        if units_consumed <= 100:
            energy_charges = units_consumed * 5
        elif units_consumed <= 300:
            energy_charges = (100 * 5) + ((units_consumed - 100) * 7)
        else:
            energy_charges = (100 * 5) + (200 * 7) + ((units_consumed - 300) * 10)
    elif customer_type == 'Commercial':
        if units_consumed <= 200:
            energy_charges = units_consumed * 8
        elif units_consumed <= 500:
            energy_charges = (200 * 8) + ((units_consumed - 200) * 12)
        else:
            energy_charges = (200 * 8) + (300 * 12) + ((units_consumed - 500) * 15)
    elif customer_type == 'Industrial':
        if units_consumed <= 500:
            energy_charges = units_consumed * 10
        elif units_consumed <= 1000:
            energy_charges = (500 * 10) + ((units_consumed - 500) * 13)
        else:
            energy_charges = (500 * 10) + (500 * 13) + ((units_consumed - 1000) * 18)

    # Display the output for each customer separately
    print("-----")
    print(f"Customer ID: {customer_id}")
    print(f"Customer Type: {customer_type}")
    print(f"Previous Units: {previous_units}")
    print(f"Current Units: {current_units}")
    print(f"Units Consumed: {units_consumed}")
    print(f"Energy Charges: Rs.{energy_charges:.2f}")
    print("-----\n")
```

OUTPUT:

```
...
Electricity Billing System - Consumer Details and Unit Consumption
...
Total Customers Processed: 12

-----
Customer ID: C001
Customer Type: Commercial
Previous Units: 906
Current Units: 1174
Units Consumed: 268
Energy Charges: Rs.2416.00
-----

-----
Customer ID: C002
Customer Type: Industrial
Previous Units: 691
Current Units: 1186
Units Consumed: 495
Energy Charges: Rs.4950.00
-----

-----
Customer ID: C003
Customer Type: Industrial
Previous Units: 758
Current Units: 1144
Units Consumed: 386
Energy Charges: Rs.3860.00
-----
```

```
...
-----
Customer ID: C004
Customer Type: Commercial
Previous Units: 435
Current Units: 816
Units Consumed: 381
Energy Charges: Rs.3772.00
-----

-----
Customer ID: C005
Customer Type: Industrial
Previous Units: 657
Current Units: 1114
Units Consumed: 457
Energy Charges: Rs.4570.00
-----

-----
Customer ID: C006
Customer Type: Commercial
Previous Units: 644
Current Units: 899
Units Consumed: 255
Energy Charges: Rs.2266.00
-----

-----
Customer ID: C007
Customer Type: Industrial
Previous Units: 100
-----
```

```

...
Customer ID: C008
Customer Type: Residential
Previous Units: 274
Current Units: 359
Units Consumed: 85
Energy Charges: Rs.425.00

-----
Customer ID: C009
Customer Type: Commercial
Previous Units: 74
Current Units: 561
Units Consumed: 487
Energy Charges: Rs.5044.00

-----
Customer ID: C010
Customer Type: Commercial
Previous Units: 170
Current Units: 282
Units Consumed: 112
Energy Charges: Rs.896.00

-----
Customer ID: C011
Customer Type: Industrial
Previous Units: 551
Current Units: 1022
Units Consumed: 471
Energy Charges: Rs.4710.00

```

```

Customer ID: C010
Customer Type: Commercial
Previous Units: 170
Current Units: 282
Units Consumed: 112
Energy Charges: Rs.896.00

-----
Customer ID: C011
Customer Type: Industrial
Previous Units: 551
Current Units: 1022
Units Consumed: 471
Energy Charges: Rs.4710.00

-----
Customer ID: C012
Customer Type: Industrial
Previous Units: 847
Current Units: 899
Units Consumed: 52
Energy Charges: Rs.520.00

```

JUSTIFICATION:

This task demonstrates real-world billing logic where charges vary by customer type and consumption slabs. Using structured tariff tables improves readability and avoids complex nested conditionals. Automatic data generation enables consistent testing without manual errors. The approach enhances scalability, allowing tariff updates without rewriting logic. Overall, it strengthens understanding of conditional logic, data handling, and clean coding practices.

Question 3:

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

Prompt:

Generate a code for Modular design using functions scenerio. The billing logic must be reusable for multiple consumers. The code uses user-defined functions to calculate energy charges and fixed charges. It should return calculated values and includes meaningful comments. Generate a efficient and readable code.

CODE:

```

import random
# --- Function Definitions ---
def calculate_energy_charges(units_consumed, customer_type):
    """
    Calculates energy charges based on units consumed and customer type.

    Args:
        units_consumed (int): The number of units consumed by the customer.
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated energy charges.
    """

    energy_charges = 0.0
    if customer_type == 'Residential':
        if units_consumed < 100:
            energy_charges = units_consumed * 5
        elif units_consumed <= 300:
            energy_charges = (100 * 5) + ((units_consumed - 100) * 7)
        else:
            energy_charges = (100 * 5) + (200 * 7) + ((units_consumed - 300) * 10)
    elif customer_type == 'Commercial':
        if units_consumed <= 200:
            energy_charges = units_consumed * 8
        elif units_consumed <= 500:
            energy_charges = (200 * 8) + ((units_consumed - 200) * 12)
        else:
            energy_charges = (200 * 8) + (300 * 12) + ((units_consumed - 500) * 15)
    elif customer_type == 'Industrial':
        """
        Calculates fixed charges based on customer type.
    """

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated fixed charges.
    """

    fixed_charges = 0.0
    if customer_type == 'Residential':
        fixed_charges = 50.0
    elif customer_type == 'Commercial':
        fixed_charges = 100.0
    elif customer_type == 'Industrial':
        fixed_charges = 200.0
    return fixed_charges
# --- Main Program Logic ---
# Generate at least 10 customer records automatically
customers_data = []
num_customers = 12 # Ensure at least 10 customers

for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'C{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })
# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Bill Calculation\n")
print(f"Total Customers Processed: {len(customers_data)}\n")

for customer in customers_data:
    # Extract customer details
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']
    customer_type = customer['customer_type']

    # Calculate units consumed
    units_consumed = current_units - previous_units

```

```

elif customer_type == 'Industrial':
    if units_consumed <= 500:
        energy_charges = units_consumed * 10
    elif units_consumed <= 1000:
        energy_charges = (500 * 10) + ((units_consumed - 500) * 13)
    else:
        energy_charges = (500 * 10) + (500 * 13) + ((units_consumed - 1000) * 18)

    return energy_charges
def calculate_fixed_charges(customer_type):
    """
    Calculates fixed charges based on customer type.

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated fixed charges.
    """

    fixed_charges = 0.0
    if customer_type == 'Residential':
        fixed_charges = 50.0
    elif customer_type == 'Commercial':
        fixed_charges = 100.0
    elif customer_type == 'Industrial':
        fixed_charges = 200.0
    return fixed_charges
# --- Main Program Logic ---
# Generate at least 10 customer records automatically
customers_data = []

```

```

# --- Main Program Logic ---
# Generate at least 10 customer records automatically
customers_data = []
num_customers = 12 # Ensure at least 10 customers

for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'C{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })
# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Bill Calculation\n")
print(f"Total Customers Processed: {len(customers_data)}\n")

for customer in customers_data:
    # Extract customer details
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']
    customer_type = customer['customer_type']

    # Calculate units consumed
    units_consumed = current_units - previous_units

```

```

current_units = customers_data[-1]['current_units']
customer_type = customers_data[-1]['customer_type']

# 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)

# Calculate Total Bill
total_bill = energy_charges + fixed_charges

# Display the output for each customer separately
print("-----")
print(f"Customer ID: {customer_id}")
print(f"Customer Type: {customer_type}")
print(f"Previous Units: {previous_units}")
print(f"Current Units: {current_units}")
print(f"Units Consumed: {units_consumed}")
print(f"Energy Charges: Rs.{energy_charges:.2f}")
print(f"Fixed Charges: Rs.{fixed_charges:.2f}")
print(f"Total Bill: Rs.{total_bill:.2f}")
print("-----\n")

```

OUTPUT:

```
Electricity Billing System - Consumer Details and Bill Calculation
```

```
*** Total Customers Processed: 12
```

```
Customer ID: C001
Customer Type: Industrial
Previous Units: 512
Current Units: 959
Units Consumed: 447
Energy Charges: Rs.4470.00
Fixed Charges: Rs.200.00
Total Bill: Rs.4670.00
```

```
Customer ID: C002
Customer Type: Commercial
Previous Units: 208
Current Units: 495
Units Consumed: 287
Energy Charges: Rs.2644.00
Fixed Charges: Rs.100.00
Total Bill: Rs.2744.00
```

```
Customer ID: C003
Customer Type: Commercial
Previous Units: 326
Current Units: 537
Units Consumed: 211
```

```
Customer ID: C007
*** Customer Type: Industrial
Previous Units: 143
Current Units: 566
Units Consumed: 423
Energy Charges: Rs.4230.00
Fixed Charges: Rs.200.00
Total Bill: Rs.4430.00
```

```
Customer ID: C008
Customer Type: Industrial
Previous Units: 12
Current Units: 499
Units Consumed: 487
Energy Charges: Rs.4870.00
Fixed Charges: Rs.200.00
Total Bill: Rs.5070.00
```

```
Customer ID: C009
Customer Type: Residential
Previous Units: 188
Current Units: 534
Units Consumed: 346
Energy Charges: Rs.2360.00
Fixed Charges: Rs.50.00
Total Bill: Rs.2410.00
```

```
Customer ID: C004
Customer Type: Commercial
Previous Units: 594
Current Units: 932
Units Consumed: 338
Energy Charges: Rs.3256.00
Fixed Charges: Rs.100.00
Total Bill: Rs.3356.00
```

```
Customer ID: C005
Customer Type: Commercial
Previous Units: 733
Current Units: 900
Units Consumed: 167
Energy Charges: Rs.1336.00
Fixed Charges: Rs.100.00
Total Bill: Rs.1436.00
```

```
Customer ID: C006
Customer Type: Residential
Previous Units: 877
Current Units: 1091
Units Consumed: 214
Energy Charges: Rs.1298.00
Fixed Charges: Rs.50.00
Total Bill: Rs.1348.00
```

```
Customer ID: C010
Customer Type: Commercial
*** Previous Units: 171
Current Units: 305
Units Consumed: 134
Energy Charges: Rs.1072.00
Fixed Charges: Rs.100.00
Total Bill: Rs.1172.00
```

```
Customer ID: C011
Customer Type: Industrial
Previous Units: 293
Current Units: 681
Units Consumed: 388
Energy Charges: Rs.3880.00
Fixed Charges: Rs.200.00
Total Bill: Rs.4080.00
```

```
Customer ID: C012
Customer Type: Industrial
Previous Units: 251
Current Units: 668
Units Consumed: 417
Energy Charges: Rs.4170.00
Fixed Charges: Rs.200.00
Total Bill: Rs.4370.00
```

```
es Terminal
```

JUSTIFICATION:

This task demonstrates modular programming by separating billing logic into reusable functions. Functions improve code readability, maintainability, and testing flexibility. Returning calculated values ensures correctness and avoids redundancy. The structure supports scalability for future tariff changes. Overall, it reflects real-world software design best practices.

Question 4

Task 4: Calculation of Additional Charges

Prompt:

Generate a code for calculation of additional charges scenario. The electricity bills should include multiple additional charges. Extend the program to calculate Fixed charges(FC), Customer Charges(CC), Electricity Duty(percentage of EC)(ED). Add Electricity duty calculation and improve billing accuracy. The code should be more efficient and readable.

CODE:

```
import random
# --- Function Definitions ---
def calculate_energy_charges(units_consumed, customer_type):
    """
    Calculates energy charges based on units consumed and customer type.

    Args:
        units_consumed (int): The number of units consumed by the customer.
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated energy charges.
    """
    energy_charges = 0.0

    if customer_type == 'Residential':
        if units_consumed <= 100:
            energy_charges = units_consumed * 5
        elif units_consumed <= 300:
            energy_charges = (100 * 5) + ((units_consumed - 100) * 7)
        else:
            energy_charges = (100 * 5) + (200 * 7) + ((units_consumed - 300) * 10)
    elif customer_type == 'Commercial':
        if units_consumed <= 200:
            energy_charges = units_consumed * 8
        elif units_consumed <= 500:
            energy_charges = (200 * 8) + ((units_consumed - 200) * 12)
        else:
            energy_charges = (200 * 8) + (300 * 12) + ((units_consumed - 500) * 15)
    elif customer_type == 'Industrial':
        if units_consumed <= 500:
            energy_charges = units_consumed * 10
        elif units_consumed <= 1000:
            energy_charges = (500 * 10) + ((units_consumed - 500) * 13)
        else:
            energy_charges = (500 * 10) + (500 * 13) + ((units_consumed - 1000) * 18)

    return energy_charges

def calculate_fixed_charges(customer_type):
    """
    Calculates fixed charges based on customer type.

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated fixed charges.
    """
    fixed_charges = 0.0

    if customer_type == 'Residential':
        fixed_charges = 100
    elif customer_type == 'Commercial':
        fixed_charges = 200
    elif customer_type == 'Industrial':
        fixed_charges = 500
```

```

"""
fixed_charges = 0.0
if customer_type == 'Residential':
    fixed_charges = 50.0
elif customer_type == 'Commercial':
    fixed_charges = 100.0
elif customer_type == 'Industrial':
    fixed_charges = 200.0
return fixed_charges

def calculate_customer_charges(customer_type):
    """
    Calculates customer charges based on customer type.

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated customer charges.
    """
    customer_charges = 0.0
    if customer_type == 'Residential':
        customer_charges = 20.0
    elif customer_type == 'Commercial':
        customer_charges = 50.0
    elif customer_type == 'Industrial':
        customer_charges = 100.0
    return customer_charges
"""

# --- Main Program Logic ---

# Generate at least 10 customer records automatically
customers_data = []
num_customers = 12 # Ensure at least 10 customers

for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })

# Electricity Duty percentage
electricity_duty_percentage = 0.05 # 5%

# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Bill Calculation\n")
print(f"Total Customers Processed: {len(customers_data)}\n")

for customer in customers_data:
    # Extract customer details
    customer_id = customer['customer_id']
    previous_units = customer['previous_units']
    current_units = customer['current_units']

    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)

    # Calculate Customer Charges using the function
    customer_charges = calculate_customer_charges(customer_type)

    # Calculate Electricity Duty
    electricity_duty = energy_charges * electricity_duty_percentage

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

    # Display the output for each customer separately
    print("-----")
    print(f"Customer ID: {customer_id}")
    print(f"Customer Type: {customer_type}")
    print(f"Previous Units: {previous_units}")
    print(f"Current Units: {current_units}")
    print(f"Units Consumed: {units_consumed}")
    print(f"Energy Charges: Rs.{energy_charges:.2f}")
    print(f"Fixed Charges: Rs.{fixed_charges:.2f}")
    print(f"Customer Charges: Rs.{customer_charges:.2f}")
    print(f"Electricity Duty: Rs.{electricity_duty:.2f}")
    print(f"Total Bill: Rs.{total_bill:.2f}")
    print("-----\n")

```

OUTPUT:

```
Electricity Billing System - Consumer Details and Bill Calculation
...
Total Customers Processed: 12

-----
Customer ID: C001
Customer Type: Residential
Previous Units: 211
Current Units: 279
Units Consumed: 68
Energy Charges: Rs.340.00
Fixed Charges: Rs.50.00
Customer Charges: Rs.20.00
Electricity Duty: Rs.17.00
Total Bill: Rs.427.00
-----

Customer ID: C002
Customer Type: Commercial
Previous Units: 739
Current Units: 880
Units Consumed: 141
Energy Charges: Rs.1128.00
Fixed Charges: Rs.100.00
Customer Charges: Rs.50.00
Electricity Duty: Rs.56.40
Total Bill: Rs.1334.40
-----
```



```
...
Customer ID: C003
Customer Type: Residential
Previous Units: 955
Current Units: 1358
Units Consumed: 403
Energy Charges: Rs.2930.00
Fixed Charges: Rs.50.00
Customer Charges: Rs.20.00
Electricity Duty: Rs.146.50
Total Bill: Rs.3146.50
-----

Customer ID: C008
Customer Type: Residential
Previous Units: 309
Current Units: 776
Units Consumed: 467
Energy Charges: Rs.3570.00
Fixed Charges: Rs.50.00
Customer Charges: Rs.20.00
Electricity Duty: Rs.178.50
Total Bill: Rs.3818.50
-----

Customer ID: C009
Customer Type: Residential
Previous Units: 709
Current Units: 795
Units Consumed: 86
Energy Charges: Rs.430.00
-----
```



```
...
Customer ID: C004
Customer Type: Commercial
Previous Units: 25
Current Units: 515
Units Consumed: 490
Energy Charges: Rs.5080.00
Fixed Charges: Rs.100.00
Customer Charges: Rs.50.00
Electricity Duty: Rs.254.00
Total Bill: Rs.5484.00
-----

Customer ID: C005
Customer Type: Industrial
Previous Units: 676
Current Units: 847
Units Consumed: 171
Energy Charges: Rs.1710.00
Fixed Charges: Rs.200.00
Customer Charges: Rs.100.00
Electricity Duty: Rs.85.50
Total Bill: Rs.2095.50
-----

Customer ID: C006
Customer Type: Industrial
Previous Units: 577
Current Units: 634
Units Consumed: 57
-----
```



```
...
Customer ID: C011
Customer Type: Industrial
Previous Units: 272
Current Units: 704
Units Consumed: 432
Energy Charges: Rs.4120.00
Fixed Charges: Rs.200.00
Customer Charges: Rs.100.00
Electricity Duty: Rs.206.00
Total Bill: Rs.4626.00
-----

Customer ID: C012
Customer Type: Residential
Previous Units: 933
Current Units: 1273
Units Consumed: 340
Energy Charges: Rs.2300.00
Fixed Charges: Rs.50.00
Customer Charges: Rs.20.00
Electricity Duty: Rs.115.00
Total Bill: Rs.2485.00
-----
```

JUSTIFICATION:

This task improves billing accuracy by including real-world additional charges such as fixed charges, customer charges, and electricity duty. Modular functions enhance readability and make the logic reusable across multiple consumers. Separating each charge into its own function simplifies maintenance and future updates. Percentage-based duty ensures realistic tax computation. Overall, the program reflects a complete and practical electricity billing system.

Question 5

Task 5: Final Bill Generation and Output Analysis

Prompt:

Generate a code for final bill generation and output analysis scenario. The final electricity bill must present all values clearly. Develop the final python application to calculate total bill: Total Bill = EC+FC+CC+ED. Display Energy charges,fixed charges, customer charges, electricity duty ,total bill amount. Analyze the program based on accuracy,readability, real world applicability. The code should be efficient and readable.

CODE:

```
✓ Task
Analyze the developed Python electricity billing application for its accuracy, readability, and real-world applicability.

✓ Develop Electricity Billing Application
Subtask:
Develop a Python application that simulates electricity bill generation. This will include generating customer data with previous and current unit readings, customer types (Residential, Commercial, Industrial), and then calculating Energy Charges (EC), Fixed Charges (FC), Customer Charges (CC), Electricity Duty (ED), and the Total Bill. The application will leverage the existing rate structures and functions, and ensure that all calculated values are displayed clearly for each customer. The code will prioritize efficiency and readability, as requested.

Reasoning: The instructions indicate that the latest code cell contains the complete electricity billing application. Therefore, executing this cell will run the application and display the generated electricity bills as per the subtask.

[1]: 
import random
# --- Function Definitions ---
def calculate_energy_charges(units_consumed, customer_type):
    """
    Calculates energy charges based on units consumed and customer type.

    Args:
        units_consumed (int): The number of units consumed by the customer.
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated energy charges.
    """
    energy_charges = 0.0

    if customer_type == 'Residential':
        if units_consumed <= 100:
            energy_charges = units_consumed * 5
        elif units_consumed <= 300:
            energy_charges = (100 * 5) + ((units_consumed - 100) * 7)
        else:
            energy_charges = (100 * 5) + (200 * 7) + ((units_consumed - 300) * 10)
    elif customer_type == 'Commercial':
        if units_consumed <= 200:
            energy_charges = units_consumed * 8
        elif units_consumed <= 500:
            energy_charges = (200 * 8) + ((units_consumed - 200) * 12)
        else:
            energy_charges = (200 * 8) + (300 * 12) + ((units_consumed - 500) * 15)
    elif customer_type == 'Industrial':
        if units_consumed <= 500:
            energy_charges = units_consumed * 10
    else:
        energy_charges = 0.0

    return energy_charges

def calculate_fixed_charges(customer_type):
    """
    Calculates fixed charges based on customer type.

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated fixed charges.
    """
    fixed_charges = 0.0

    if customer_type == 'Residential':
        fixed_charges = 50.0
    elif customer_type == 'Commercial':
        fixed_charges = 100.0
    elif customer_type == 'Industrial':
        fixed_charges = 200.0

    return fixed_charges

def calculate_customer_charges(customer_type):
    """
    Calculates customer charges based on customer type.

    Args:
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The calculated customer charges.
    """
    customer_charges = 0.0

    if customer_type == 'Residential':
        customer_charges = 0.05 * (fixed_charges + energy_charges)
    elif customer_type == 'Commercial':
        customer_charges = 0.1 * (fixed_charges + energy_charges)
    elif customer_type == 'Industrial':
        customer_charges = 0.15 * (fixed_charges + energy_charges)

    return customer_charges

def calculate_total_bill(customer_id, previous_units, current_units, customer_type):
    """
    Calculates the total bill for a given customer.

    Args:
        customer_id (str): The unique identifier for the customer.
        previous_units (int): The previous unit reading.
        current_units (int): The current unit reading.
        customer_type (str): The type of the customer (Residential, Commercial, Industrial).

    Returns:
        float: The total bill amount.
    """
    energy_charges = calculate_energy_charges(current_units - previous_units, customer_type)
    fixed_charges = calculate_fixed_charges(customer_type)
    customer_charges = calculate_customer_charges(customer_type)
    electricity_duty = 0.05 * (fixed_charges + energy_charges)

    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

    return total_bill

# --- Main Program Logic ---
# Generate at least 10 customer records automatically
customers_data = []
num_customers = 12 # Ensure at least 10 customers

for i in range(1, num_customers + 1):
    previous_units = random.randint(0, 1000)
    current_units = previous_units + random.randint(50, 500) # Ensure CU > PU
    customer_type = random.choice(['Residential', 'Commercial', 'Industrial'])

    customers_data.append({
        'customer_id': f'{i:03d}',
        'previous_units': previous_units,
        'current_units': current_units,
        'customer_type': customer_type
    })

# Electricity Duty percentage
electricity_duty_percentage = 0.05 # 5%

# Process each customer record and display details separately
print("Electricity Billing System - Consumer Details and Bill Calculation\n")
print(f"Total Customers Processed: {len(customers_data)}\n")

for customer in customers_data:
    # Extract customer details
    customer_id = customer['customer_id']
    previous_units = customer['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)

    # Calculate Customer Charges using the function
    customer_charges = calculate_customer_charges(customer_type)

    # Calculate Electricity Duty
    electricity_duty = energy_charges * electricity_duty_percentage

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

    # Display the output for each customer separately
    print("-----")
    print(f"Customer ID: {customer_id}")
    print(f"Customer Type: {customer_type}")
    print(f"Previous Units: {previous_units}")
    print(f"Current Units: {current_units}")
    print(f"Units Consumed: {units_consumed}")
    print(f"Energy Charges: Rs.{energy_charges:.2f}")
    print(f"Fixed Charges: Rs.{fixed_charges:.2f}")
    print(f"Customer Charges: Rs.{customer_charges:.2f}")
    print(f"Electricity Duty: Rs.{electricity_duty:.2f}")
    print(f"Total Bill: Rs.{total_bill:.2f}")
    print("-----\n")

```

OUTPUT:

Electricity Billing System - Consumer Details and Bill Calculation	
*** Total Customers Processed: 12	
----- Customer ID: C001 Customer Type: Industrial Previous Units: 486 Current Units: 746 Units Consumed: 260 Energy Charges: Rs.2600.00 Fixed Charges: Rs.200.00 Customer Charges: Rs.100.00 Electricity Duty: Rs.130.00 Total Bill: Rs.3030.00 ----- Customer ID: C002 Customer Type: Commercial Previous Units: 904 Current Units: 1395 Units Consumed: 491 Energy Charges: Rs.5092.00 Fixed Charges: Rs.100.00 Customer Charges: Rs.50.00 Electricity Duty: Rs.254.60 Total Bill: Rs.5496.60 ----- Customer ID: C003	
	----- Fixed Charges: Rs.200.00 Customer Charges: Rs.100.00 Electricity Duty: Rs.60.50 Total Bill: Rs.1570.50 ----- ----- Customer ID: C011 Customer Type: Industrial Previous Units: 451 Current Units: 579 Units Consumed: 128 Energy Charges: Rs.1280.00 Fixed Charges: Rs.200.00 Customer Charges: Rs.100.00 Electricity Duty: Rs.64.00 Total Bill: Rs.1644.00 ----- ----- Customer ID: C012 Customer Type: Commercial Previous Units: 685 Current Units: 1051 Units Consumed: 366 Energy Charges: Rs.3592.00 Fixed Charges: Rs.100.00 Customer Charges: Rs.50.00 Electricity Duty: Rs.179.60 Total Bill: Rs.3921.60

Analyze Billing Program

Subtask:

Provide a detailed textual analysis of the developed electricity billing program covering accuracy, readability, and real-world applicability.

Analysis of the Electricity Billing Program

1. Accuracy

The calculation logic within the `calculate_energy_charges`, `calculate_fixed_charges`, `calculate_customer_charges` functions, and for `electricity_duty` appears to be accurate based on the provided rate structures and percentages.

- **Energy Charges:** The tiered billing for 'Residential', 'Commercial', and 'Industrial' customer types is correctly implemented. For example, for a Residential customer consuming 336 units, the calculation is $(100 * 5) + (200 * 7) + (36 * 10) = 500 + 1400 + 360 = 2260.00$, which matches the output. Similarly, for a Commercial customer consuming 491 units, it's $(200 * 8) + (291 * 12) = 1600 + 3492 = 5092.00$, also matching the output. An Industrial customer consuming 260 units is billed at $260 * 10 = 2600.00$, which is also correct.
 - **Fixed Charges:** The fixed charges (Residential: Rs.50.00, Commercial: Rs.100.00, Industrial: Rs.200.00) are applied correctly based on the `customer_type`.
 - **Customer Charges:** The customer charges (Residential: Rs.20.00, Commercial: Rs.50.00, Industrial: Rs.100.00) are also correctly applied based on the `customer_type`.
- applied based on the `customer_type`.
- **Electricity Duty:** The `electricity_duty` is consistently calculated as 5% of the `energy_charges`, which aligns with the `electricity_duty_percentage = 0.05` defined in the code.
 - **Total Bill:** The `total_bill` is correctly summed as `energy_charges + fixed_charges + customer_charges + electricity_duty`.

The manual cross-verification of several examples from the output of the latest execution confirms the computational accuracy of the program against its defined rules.

2. Readability

The program demonstrates excellent readability due to several factors:

- **Code Structure:** The code is modularized effectively through the use of distinct functions (`calculate_energy_charges`, `calculate_fixed_charges`, `calculate_customer_charges`). This separation of concerns makes the code easier to understand, test, and maintain.
- **Variable Names:** All variables are descriptively named (e.g., `previous_units`, `current_units`, `units_consumed`, `customer_type`, `energy_charges`, `electricity_duty_percentage`, `total_bill`). This clear naming convention minimizes ambiguity and improves comprehension.
- **Comments and Docstrings:** Each function includes a comprehensive docstring explaining its purpose, arguments, and return values. This is a best practice for documenting Python code. Additionally, there are inline comments where appropriate, such as for the `electricity_duty_percentage`, which further aids understanding.
- **Logical Flow and Organization:** The main program logic is straightforward. It begins with data generation, followed by function

3. Real-World Applicability

This Python application serves as a good **simulation** of an electricity billing system, providing a functional foundation for understanding basic billing logic. However, its real-world applicability is somewhat limited due to simplifications:

- **Customer Data Generation:** The customer data (previous units, current units, customer type) is randomly generated. In a real-world scenario, this data would be sourced from actual meter readings, utility databases, and customer records. While suitable for a simulation, it doesn't reflect the complexities of data acquisition and management in practice.
- **Simplicity of Charge Structure:** The current charge structure includes Energy Charges, Fixed Charges, Customer Charges, and Electricity Duty. While these are common elements, actual electricity bills often involve:
 - **Additional Taxes/Surcharges:** Beyond a single electricity duty, there might be state/local taxes, environmental surcharges, regulatory charges, or fuel adjustment charges.
 - **Demand Charges:** Especially for commercial and industrial customers, charges based on peak power demand (kW) are crucial.
 - **Minimum Charges:** A base fee applied even if consumption is very low or zero.
 - **Seasonal or Time-of-Use (TOU) Rates:** Rates that vary depending on the time of day (peak/off-peak) or season (summer/winter) are common.
 - **Late Payment Fees:** Penalties for overdue bills.
 - **Subsidies or Rebates:** Specific customer segments might receive financial assistance.
- **Billing Cycles:** The simulation processes all generated customers at once. Real-world systems manage staggered billing cycles throughout the month or year, and historical consumption data is often crucial for analysis and billing adjustments.
- **Absence of Other Features:** A comprehensive real-world billing system would include features such as customer account

Toggle Gemini

Potential Improvements or Extensions for Real-World Scenarios:

1. **External Data Integration:** Read customer and usage data from external files (e.g., CSV, JSON) or a database (e.g., SQLite, PostgreSQL) rather than random generation.
2. **Expanded Charge Categories:** Introduce more charge types, such as demand charges, additional taxes, or time-of-use rates.
3. **Configurable Rates:** Store rate structures and fixed charges in an external configuration file (e.g., JSON, YAML) or a database table, allowing for easier updates without code modification.
4. **Error Handling and Validation:** Implement robust error checking for invalid inputs, such as negative unit consumption or non-existent customer types.
5. **Reporting and Analytics:** Generate aggregated reports (e.g., total revenue per customer type, average consumption, high-consumption alerts) and potentially visualize data.
6. **User Interface:** Develop a simple command-line interface (CLI) or a basic graphical user interface (GUI) for interactive input and display.
7. **Data Persistence:** Save calculated bills and customer data to a persistent storage medium (e.g., a file, a database) for historical tracking.

In summary, the program is an excellent didactic tool for illustrating the core logic of electricity billing. To transition it into a robust real-world application, it would require significant enhancements in data handling, complexity of rate structures, error management, and feature set.

Final Task

Subtask:

Summarize the developed electricity billing application, including its functionality, the analysis findings regarding accuracy, readability, and real-world applicability, and any key takeaways.

Summary:

Q&A

The developed Python electricity billing application was analyzed for its accuracy, readability, and real-world applicability.

Data Analysis Key Findings

- **Functionality:** The application successfully generates customer data for 12 customers, calculates units consumed, and computes Energy Charges (EC), Fixed Charges (FC), Customer Charges (CC), Electricity Duty (ED), and the Total Bill, displaying all details clearly for each customer.
- **Accuracy:** All calculations, including the tiered Energy Charges based on customer type and consumption, fixed charges, customer charges, and a 5% electricity duty on energy charges, were confirmed to be accurate through manual cross-verification of several examples.
- **Readability:** The code demonstrates excellent readability due to its modular structure with distinct functions, descriptive variable names, comprehensive docstrings, and a clear, well-organized logical flow.

Energy Charges (EC), Fixed Charges (FC), Customer Charges (CC), Electricity Duty (ED), and the Total Bill, displaying all details clearly for each customer.

- **Accuracy:** All calculations, including the tiered Energy Charges based on customer type and consumption, fixed charges, customer charges, and a 5% electricity duty on energy charges, were confirmed to be accurate through manual cross-verification of several examples.
- **Readability:** The code demonstrates excellent readability due to its modular structure with distinct functions, descriptive variable names, comprehensive docstrings, and a clear, well-organized logical flow.
- **Real-World Applicability:** While serving as a good simulation for understanding basic billing logic, the application has limited real-world applicability due to simplifications such as randomly generated customer data, a simple charge structure (lacking elements like demand charges, seasonal rates, or additional taxes), and the absence of features like external data integration, robust error handling, or persistent data storage.

Insights or Next Steps

- The current application effectively serves as a didactic tool for demonstrating core electricity billing logic.
- To transition into a robust real-world system, enhancements are needed in data handling (e.g., integrating external data sources), rate structure complexity (e.g., adding demand charges, time-of-use rates), error management, and feature set (e.g., reporting, user interface, data persistence).

JUSTIFICATION:

This task ensures complete and accurate electricity bill computation using realistic billing components. Modular functions improve readability, reusability, and maintainability of the program. Clear output formatting enhances user understanding of individual charges. Automated data generation enables efficient testing without manual errors. The design closely reflects real-world electricity billing systems.