

ASSIGNMENT – 3.3

2303A51355

Batch-10

Task-1

Prompt: I want an application for the TGNPDCL electricity bill payment system generation that must collect accurate consumer data, that needs previous units, current units, and the type of customer, code in python language.

Code :

```
def calculate_bill(previous_units, current_units, customer_type):
    units_consumed = current_units - previous_units
    if customer_type.lower() == 'residential':
        if units_consumed <= 100:
            bill_amount = units_consumed * 1.5
        elif units_consumed <= 300:
            bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5
        else:
            bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
    elif customer_type.lower() == 'commercial':
        if units_consumed <= 100:
            bill_amount = units_consumed * 2.0
        elif units_consumed <= 300:
            bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5
        else:
            bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
    else:
        raise ValueError("Invalid customer type. Please enter 'residential' or 'commercial'.")
    return bill_amount

# Collecting user input
consumer_id = input("Enter Consumer ID: ")
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Customer Type (residential/commercial): ")
```

Calculating bill

try:

```
bill_amount = calculate_bill(previous_units, current_units, customer_type)
```

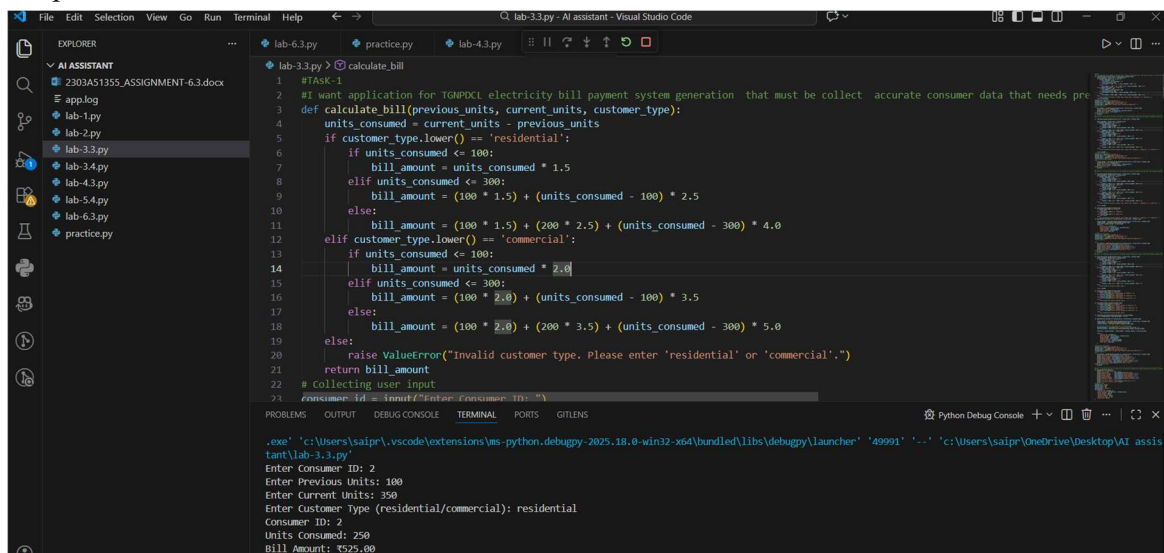
```
print(f"Consumer ID: {consumer_id}") print(f"Units Consumed:
```

```
{current_units - previous_units}") print(f"Bill Amount:
```

```
{bill_amount:.2f}") except ValueError as e:
```

```
print(e)
```

Output :



```
1 #TASK-1
2 #I want application for TGMDC electricity bill payment system generation that must be collect accurate consumer data that needs pre
3 def calculate_bill(previous_units, current_units, customer_type):
4     units_consumed = current_units - previous_units
5     if customer_type.lower() == 'residential':
6         if units_consumed <= 100:
7             bill_amount = units_consumed * 1.5
8         elif units_consumed <= 300:
9             bill_amount = (100 * 1.5) + (units_consumed - 100) * 2.5
10        else:
11            bill_amount = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
12    elif customer_type.lower() == 'commercial':
13        if units_consumed <= 100:
14            bill_amount = units_consumed * 2.0
15        elif units_consumed <= 300:
16            bill_amount = (100 * 2.0) + (units_consumed - 100) * 3.5
17        else:
18            bill_amount = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
19    else:
20        raise ValueError("Invalid customer type. Please enter 'residential' or 'commercial'.")
21    return bill_amount
22 # Collecting user input
23 consumer_id = input("Enter Consumer ID: ")
24
25 .exe" 'c:\Users\sai\r\.vscode\extensions\ms-python.debugpy-2025.18.0-win32-x64\libs\debugpy\launcher' '49991' '-' 'c:\Users\sai\r\OneDrive\Desktop\AI assis
tant\lab-3.3.py'
Enter Consumer ID: 2
Enter Previous Units: 100
Enter Current Units: 350
Enter Customer Type (residential/commercial): residential
Consumer ID: 2
Units Consumed: 250
Bill Amount: ₹525.00
```

Code Analysis :

- ☐ The program uses a function calculate_bill() to separate billing logic from user input.
- ☐ Units consumed are calculated by subtracting previous units from current units.
- ☐ Nested if-elif conditions apply slab-based tariff rates.
- ☐ Customer type is normalized using .lower() to avoid case mismatch errors.
- ☐ Exception handling ensures invalid customer types are handled safely.

Task-2

Prompt: find code from Task-1 and extend it to find energy charges that were based on domestic, commercial, and industrial consumers in optimised condition statements in Python language.

Code :

```
def calculate_energy_charges(previous_units, current_units, consumer_type):

    units_consumed = current_units - previous_units
    if consumer_type.lower() == 'domestic':
        if
units_consumed <= 100:
            charges = units_consumed * 1.2
        elif units_consumed <= 300:
            charges = (100 * 1.2) + (units_consumed - 100) * 2.0
        else:
            charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
    elif consumer_type.lower() == 'commercial':
        if units_consumed <= 100:
            charges = units_consumed * 2.5
        elif units_consumed <= 300:
            charges = (100 * 2.5) + (units_consumed - 100) * 4.0
        else:
            charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
    elif consumer_type.lower() == 'industrial':
        if units_consumed <= 100:
            charges = units_consumed * 3.0
        elif units_consumed <= 300:
            charges = (100 * 3.0) + (units_consumed - 100) * 5.0
        else:
            charges = (100 * 3.0) + (200 * 5.0) + (units_consumed - 300) * 7.5
        else:
            raise ValueError("Invalid consumer type. Please enter 'domestic', 'commercial', or
'industrial'.")
```

```

    return charges # Collecting user input consumer_id = input("Enter Consumer ID:
") previous_units = int(input("Enter Previous Units: ")) current_units =
int(input("Enter Current Units: ")) consumer_type = input("Enter Consumer Type
(domestic/commercial/industrial): ")
# Calculating energy charges
try:
    energy_charges = calculate_energy_charges(previous_units, current_units,
consumer_type)    print(f'Consumer ID: {consumer_id}')
print(f'Units Consumed: {current_units - previous_units}')
print(f'Energy Charges: ₹{energy_charges:.2f}') except
ValueError as e:
    print(e)

```

Output :

```

35 #=====
36 any TASK-2 is find code from Task-1 and extend it to find energy charges that was based on domestic,commercial,industrial consumers in
37 def calculate_energy_charges(previous_units, current_units, consumer_type):
38     units_consumed = current_units - previous_units
39     if consumer_type.lower() == 'domestic':
40         if units_consumed <= 100:
41             charges = units_consumed * 1.2
42         elif units_consumed <= 300:
43             charges = (100 * 1.2) + (units_consumed - 100) * 2.0
44         else:
45             charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
46     elif consumer_type.lower() == 'commercial':
47         if units_consumed <= 100:
48             charges = units_consumed * 2.5
49         elif units_consumed <= 300:
50             charges = (100 * 2.5) + (units_consumed - 100) * 4.0
51         else:
52             charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
53     elif consumer_type.lower() == 'industrial':
54         if units_consumed <= 100:
55             charges = units_consumed * 3.0
56         elif units_consumed <= 300:

```

```

thon.debugpy-2025.18.0-win32-x64\bundled\libs\debugpy\launcher' '63378' '-' 'c:\Users\saipe\OneDrive\Desktop\AI_assistant\lab-3.3.py'
Enter Consumer ID: 01
Enter Previous Units: 200
Enter Current Units: 500
Enter Consumer Type (domestic/commercial/industrial): domestic
Consumer ID: 01
Units Consumed: 300
Energy Charges: ₹520.00

```

Code Analysis :

- ☐ A single function, calculate_energy_charges() handles all consumer categories.
- ☐ Slab-based billing is implemented using structured conditional blocks.
- ☐ Logical grouping avoids redundant calculations.
- ☐ Function returns computed charges for further processing.
- ☐ Error handling improves robustness against invalid inputs.

Task-3

Prompt: find billing logic that must be reusable for multiple consumers, and that uses user-defined functions to calculate energy charges, fixed charges, and that should return values including meaningful comments in Python.

Code :

```
def calculate_energy_charges(previous_units, current_units, consumer_type):
    units_consumed = current_units - previous_units
    if consumer_type.lower() == 'domestic':
        if units_consumed <= 100:
            charges = units_consumed * 1.2
        elif units_consumed <= 300:
            charges = (100 * 1.2) + (units_consumed - 100) * 2.0
        else:
            charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
    elif consumer_type.lower() == 'commercial':
        if units_consumed <= 100:
            charges = units_consumed * 2.5
        elif units_consumed <= 300:
            charges = (100 * 2.5) + (units_consumed - 100) * 4.0
        else:
            charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
    elif consumer_type.lower() == 'industrial':
        if units_consumed <= 100:
            charges = units_consumed * 3.0
        elif units_consumed <= 300:
            charges = (100 * 3.0) + (units_consumed - 100) * 5.0
        else:
            charges = (100 * 3.0) + (200 * 5.0) + (units_consumed - 300) * 7.5
    else:
        raise ValueError("Invalid consumer type. Please enter 'domestic', 'commercial', or 'industrial'.")

    return charges
```

```

def calculate_fixed_charges(consumer_type):
    if consumer_type.lower() == 'domestic':
        return 50.0    elif consumer_type.lower()
    == 'commercial':    return 100.0    elif
    consumer_type.lower() == 'industrial':
        return 150.0
    else:

        raise ValueError("Invalid consumer type. Please enter 'domestic', 'commercial', or
'industrial'.")
def generate_bill(consumer_id, previous_units, current_units,
consumer_type):

    energy_charges = calculate_energy_charges(previous_units, current_units,
consumer_type)    fixed_charges =
    calculate_fixed_charges(consumer_type)    total_bill =
    energy_charges + fixed_charges    return {
        "Consumer ID": consumer_id,
        "Units Consumed": current_units - previous_units,
        "Energy Charges": energy_charges,
        "Fixed Charges": fixed_charges,
        "Total Bill": total_bill
    }
# Collecting user input
consumer_id = input("Enter Consumer ID: ")
previous_units = int(input("Enter Previous Units: "))
current_units =
int(input("Enter Current Units: "))
consumer_type = input("Enter Consumer Type
(domestic/commercial/industrial): ")
# Generating bill
try:
    bill_details = generate_bill(consumer_id, previous_units, current_units,

```

```

consumer_type) print(f'Consumer ID: {bill_details['Consumer
ID']})") print(f'Units Consumed: {bill_details['Units
Consumed']})") print(f'Energy Charges: ₹{bill_details['Energy
Charges']:.2f}')") print(f'Fixed Charges: ₹{bill_details['Fixed
Charges']:.2f}')") print(f'Total Bill: ₹{bill_details['Total
Bill']:.2f}')") except ValueError as e:

    print(e)

```

Output :

```

77 """
78 #~
79 #my Task-3 is to find billing logic must be reusable for multiple consumers and that uses user defined functions to calculate energy ch
80
81 def calculate_energy_charges(previous_units, current_units, consumer_type):
82     units_consumed = current_units - previous_units
83     if consumer_type.lower() == 'domestic':
84         if units_consumed <= 100:
85             charges = units_consumed * 1.2
86         elif units_consumed <= 300:
87             charges = (100 * 1.2) + (units_consumed - 100) * 2.0
88         else:
89             charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
90     elif consumer_type.lower() == 'commercial':
91         if units_consumed <= 100:
92             charges = units_consumed * 2.5
93         elif units_consumed <= 300:
94             charges = (100 * 2.5) + (units_consumed - 100) * 4.0
95         else:
96             charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
97     elif consumer_type.lower() == 'industrial':
98         if units_consumed <= 100:
99             charges = units_consumed * 3.0
100
101     return charges
102
103 def generate_bill(consumer_id, previous_units, current_units, consumer_type):
104     energy_charges = calculate_energy_charges(previous_units, current_units, consumer_type)
105     fixed_charges = 50
106     total_bill = energy_charges + fixed_charges
107     return {
108         'Consumer ID': consumer_id,
109         'Units Consumed': current_units - previous_units,
110         'Energy Charges': energy_charges,
111         'Fixed Charges': fixed_charges,
112         'Total Bill': total_bill
113     }
114
115 if __name__ == '__main__':
116     consumer_id = input("Enter Consumer ID: ")
117     previous_units = int(input("Enter Previous Units: "))
118     current_units = int(input("Enter Current Units: "))
119     consumer_type = input("Enter Consumer Type (domestic/commercial/industrial): ")
120     bill_details = generate_bill(consumer_id, previous_units, current_units, consumer_type)
121     print(f"Consumer ID: {bill_details['Consumer ID']}")
122     print(f"Units Consumed: {bill_details['Units Consumed']}")
123     print(f"Energy Charges: ₹{bill_details['Energy Charges']:.2f}")
124     print(f"Fixed Charges: ₹{bill_details['Fixed Charges']:.2f}")
125     print(f"Total Bill: ₹{bill_details['Total Bill']:.2f}")
126
127 except ValueError as e:
128     print(e)

```

Users\saige\OneDrive\Desktop\AI assistant\lab-3.3.py
Enter Consumer ID: 03
Enter Previous Units: 200
Enter Current Units: 400
Enter Consumer Type (domestic/commercial/industrial): domestic
Consumer ID: 03
Units Consumed: 200
Energy Charges: ₹320.00
Fixed Charges: ₹50.00
Total Bill: ₹370.00

Code Analysis :

- ☐ Code is modularised using multiple user-defined functions.
- ☐ Energy charges and fixed charges are calculated independently.
- ☐ generate_bill() integrates all charge components into one structure.
- ☐ Dictionary return type improves readability and structured output.
- ☐ Design supports reuse for multiple consumers efficiently.

Task-4

Prompt: generate an electricity bill including multiple additional charges like fixed charges, customer charges, percentage of electricity duty, and duty calculation by improving accuracy.

Code :

```
def calculate_energy_charges(previous_units, current_units, consumer_type):  
    units_consumed = current_units - previous_units  
    if consumer_type.lower() == 'domestic':  
        if  
units_consumed <= 100:  
            charges = units_consumed * 1.2  
        elif units_consumed <= 300:  
            charges = (100 * 1.2) + (units_consumed - 100) * 2.0  
        else:  
            charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5  
    elif consumer_type.lower() == 'commercial':  
        if units_consumed <=  
100:  
            charges = units_consumed * 2.5  
        elif units_consumed <= 300:  
            charges = (100 * 2.5) + (units_consumed - 100) * 4.0  
        else:  
            charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0  
    elif consumer_type.lower() == 'industrial':  
        if units_consumed <=  
100:  
            charges = units_consumed * 3.5  
        elif units_consumed <= 300:  
            charges = (100 * 3.5) + (units_consumed - 100) * 5.5  
        else:  
            charges = (100 * 3.5) + (200 * 5.5) + (units_consumed - 300) * 7.5  
        else:
```



```

        raise ValueError("Invalid consumer type.")

    return charges

def calculate_fixed_charges(consumer_type):
    if consumer_type.lower() == 'domestic':
        return float(input("Enter Fixed Charges for Domestic: "))
    elif consumer_type.lower() == 'commercial':
        return float(input("Enter Fixed Charges for Commercial: "))
    elif consumer_type.lower() == 'industrial':
        return float(input("Enter Fixed Charges for Industrial: "))
    else:
        raise ValueError("Invalid consumer type.")

def calculate_customer_charges(consumer_type):
    if consumer_type.lower() == 'domestic':
        return float(input("Enter Customer Charges for Domestic: "))
    elif consumer_type.lower() == 'commercial':
        return float(input("Enter Customer Charges for Commercial: "))
    elif consumer_type.lower() == 'industrial':
        return float(input("Enter Customer Charges for Industrial: "))
    else:
        raise ValueError("Invalid consumer type.")

def calculate_electricity_duty(energy_charges, duty_percentage):
    return energy_charges * duty_percentage / float(1)

def generate_bill(consumer_id, previous_units, current_units, consumer_type):

```

```

    energy_charges = calculate_energy_charges(previous_units, current_units,
consumer_type)    fixed_charges = calculate_fixed_charges(consumer_type)
customer_charges = calculate_customer_charges(consumer_type)

    # Calculate electricity duty based on a fixed percentage    duty_percentage =
float(input("Enter Electricity Duty Percentage: "))    electricity_duty =
calculate_electricity_duty(energy_charges, duty_percentage)

    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

    return {
        "Consumer ID": consumer_id,
        "Units Consumed": current_units - previous_units,
        "Energy Charges": energy_charges,
        "Fixed Charges": fixed_charges,
        "Customer Charges": customer_charges,
        "Electricity Duty": electricity_duty,
        "Total Bill": total_bill
    }

```

```

# Collecting user input consumer_id = input("Enter Consumer ID: ")
previous_units = int(input("Enter Previous Units: "))    current_units =
int(input("Enter Current Units: ")) consumer_type = input("Enter Consumer Type
(domestic/commercial/industrial): ")

# Generating bill

try:
    bill_details = generate_bill(consumer_id, previous_units, current_units, consumer_type)
    print(f"Consumer ID: {bill_details['Consumer ID']}")    print(f"Units Consumed:
{bill_details['Units Consumed']}")    print(f"Energy Charges: ₹{bill_details['Energy

```

```
Charges']:.2f}") print(f"Fixed Charges: ₹{bill_details['Fixed Charges']:.2f}")
```

```
print(f"Customer Charges: ₹{bill_details['Customer Charges']:.2f}")
```

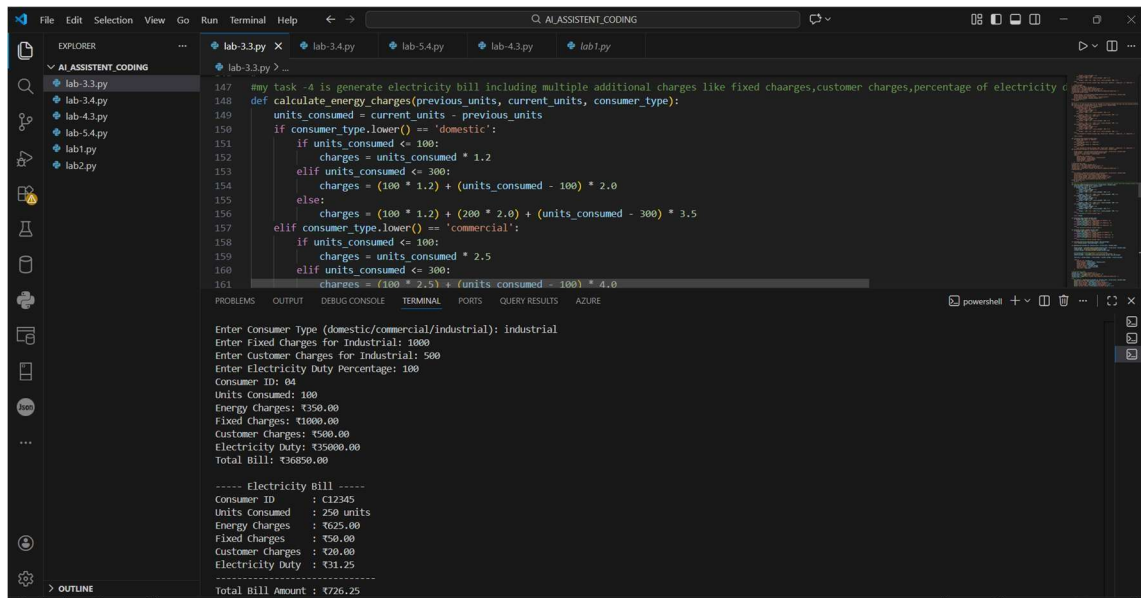
```
print(f"Electricity Duty: ₹{bill_details['Electricity Duty']:.2f}")
```

```
print(f"Total Bill: ₹{bill_details['Total Bill']:.2f}") except
```

ValueError as e:

```
print(e)
```

Output :



```
147 #my task -4 is generate electricity bill including multiple additional charges like fixed chaarges,customer charges,percentage of electricity c
148 def calculate_energy_charges(previous_units, current_units, consumer_type):
149     units_consumed = current_units - previous_units
150     if consumer_type.lower() == 'domestic':
151         if units_consumed <= 100:
152             charges = units_consumed * 1.2
153         elif units_consumed <= 300:
154             charges = (100 * 1.2) + (units_consumed - 100) * 2.0
155         else:
156             charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
157     elif consumer_type.lower() == 'commercial':
158         if units_consumed <= 100:
159             charges = units_consumed * 2.5
160         elif units_consumed <= 300:
161             charges = (100 * 2.5) + (units_consumed - 100) * 4.0
162     return charges

Enter Consumer Type (domestic/commercial/industrial): Industrial
Enter Fixed Charges for Industrial: 1000
Enter Customer Charges for Industrial: 500
Enter Electricity Duty Percentage: 100
Consumer ID: 04
Units Consumed: 100
Energy Charges: ₹350.00
Fixed Charges: ₹1000.00
Customer Charges: ₹500.00
Electricity Duty: ₹35000.00
Total Bill: ₹36850.00

----- Electricity Bill -----
Consumer ID : C12345
Units Consumed : 250 units
Energy Charges : ₹625.00
Fixed Charges : ₹50.00
Customer Charges : ₹20.00
Electricity Duty : ₹31.25
Total Bill Amount : ₹726.25
```

Code Analysis :

- ☐ Additional charge components are added through dedicated functions.
- ☐ Electricity duty is computed as a percentage of energy charges.
- ☐ Dynamic user input improves billing accuracy and flexibility.
- ☐ Functions maintain separation of concerns for clarity.
- ☐ The total bill aggregates all computed components systematically.

Task 5

Prompt: generate the final bill of electricity, including all charges, with proper formatting and display in Python language.

```

code : ef

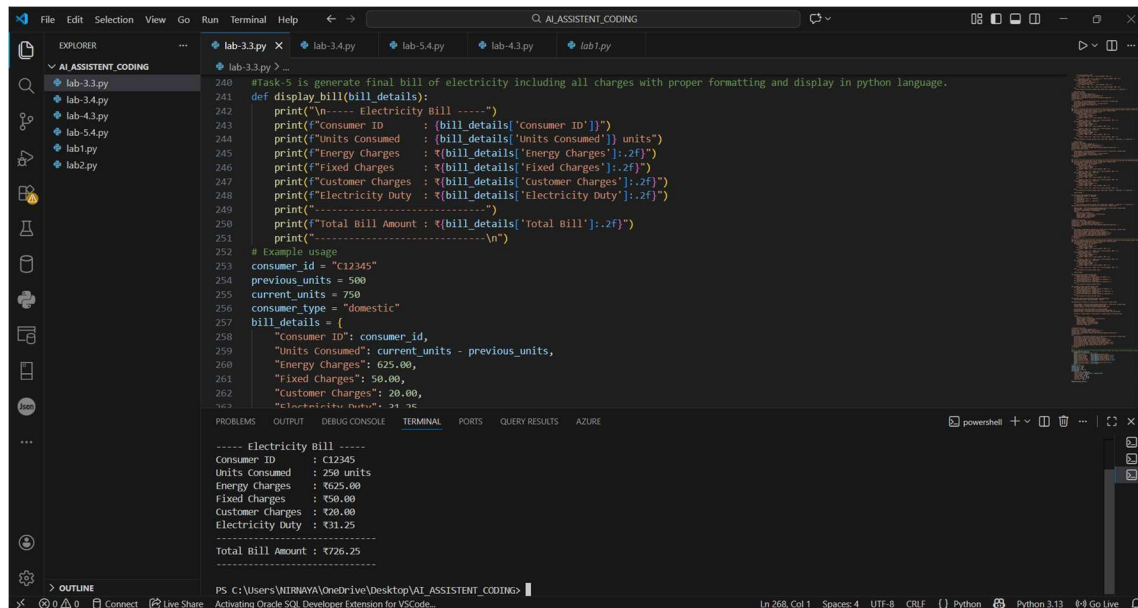
display_bill(bill_details):

    print("\n----- Electricity Bill -----")    print(f"Consumer
ID      : {bill_details['Consumer ID']}")    print(f"Units
Consumed   : {bill_details['Units Consumed']} units")
    print(f"Energy Charges   : ₹{bill_details['Energy
Charges']:.2f}")    print(f"Fixed Charges   :
₹{bill_details['Fixed Charges']:.2f}")    print(f"Customer
Charges : ₹{bill_details['Customer Charges']:.2f}")
    print(f"Electricity Duty : ₹{bill_details['Electricity
Duty']:.2f}")    print("-----")
    print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")
    print("-----\n")

# Example usage consumer_id
= "C12345"    previous_units =
500    current_units = 750
consumer_type = "domestic"
bill_details = {
    "Consumer ID": consumer_id,
    "Units Consumed": current_units - previous_units,
    "Energy Charges": 625.00,
    "Fixed Charges": 50.00,
    "Customer Charges": 20.00,
    "Electricity Duty": 31.25,
    "Total Bill": 726.25
} display_bill(bill_details)

```

Output :



```
240 #task 5 is generate final bill of electricity including all charges with proper formatting and display in python language.
241 def display_bill(bill_details):
242     print("\n----- Electricity Bill -----")
243     print(f"Consumer ID      : {bill_details['Consumer ID']}")
244     print(f"Units Consumed      : {bill_details['Units Consumed']} units")
245     print(f"Energy Charges       : ₹{bill_details['Energy Charges']:.2f}")
246     print(f"Fixed Charges        : ₹{bill_details['Fixed Charges']:.2f}")
247     print(f"Customer Charges     : ₹{bill_details['Customer Charges']:.2f}")
248     print(f"Electricity Duty      : ₹{bill_details['Electricity Duty']:.2f}")
249     print("-----")
250     print(f"Total Bill Amount : ₹{bill_details['Total bill']:.2f}")
251     print("\n-----\n")
252 # Example usage
253 consumer_id = "C12345"
254 previous_units = 500
255 current_units = 750
256 consumer_type = "domestic"
257 bill_details = {
258     "Consumer ID": consumer_id,
259     "Units Consumed": current_units - previous_units,
260     "Energy Charges": 625.00,
261     "Fixed Charges": 50.00,
262     "Customer Charges": 20.00,
263     "Electricity Duty": 21.25
264 }

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS AZURE
----- Electricity Bill -----
Consumer ID      : C12345
Units Consumed      : 250 units
Energy Charges       : ₹625.00
Fixed Charges        : ₹50.00
Customer Charges     : ₹20.00
Electricity Duty      : ₹21.25
-----
Total Bill Amount : ₹726.25
-----

PS C:\Users\MIIRNAYA\OneDrive\Desktop\VAI_ASSISTENT_CODING>
```

Code Analysis :

- ☐ Display logic is isolated in the display_bill() function.
- ☐ Formatted printing ensures professional bill presentation.
- ☐ Uses dictionary keys to access bill components safely.
- ☐ Separation of calculation and presentation improves maintainability.
- ☐ Supports easy modification for real-world billing systems.