

ASSIGNMENT – 3.3

2303A51060

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 langauge.

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 :

```

File Edit Selection View Go Run Terminal Help ← → Q AI_ASSISTANT_CODING
EXPLORER AI_ASSISTANT_CODING lab-3.3.py lab-3.4.py lab-4.3.py lab-5.4.py lab1.py lab2.py
Q AI_ASSISTANT_CODING
lab-3.3.py > ...
1 #TASK-1
2 #I want application for TGNPCL electricity bill payment system generation that must be collect accurate consumer data that needs previous unit consumed and current unit consumed
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
22    return bill_amount
23 # collecting user input
24 consumer_id = input("Enter Consumer ID: ")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS AZURE

```

PS C:\Users\WIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING> & C:\Users\WIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/WIRNAYA/Desktop/AI_ASSISTANT_CODING/lab-3.3.py
Enter Consumer ID: 1
Enter Previous Units: 200
Enter Current Units: 350
Enter Customer Type (residential/commercial): residential
Consumer ID: 1
Units Consumed: 150
Bill Amount: ₹775.00

```

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
    else:
        charges = 0.0

    return charges
```

```

    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 :

The screenshot shows a VS Code interface with the following details:

- File Explorer:** Shows files: lab-3.3.py, lab-3.4.py, lab-4.3.py, lab-5.4.py, lab1.py, and lab2.py.
- Code Editor:** Displays a Python script named lab-3.3.py. The code defines a function calculate_energy_charges that takes previous_units, current_units, and consumer_type as parameters. It uses conditional blocks to calculate charges based on consumer type (domestic, commercial, industrial) and units consumed (up to 100, between 100 and 300, or more than 300).
- Terminal:** Shows the command line output of running the script. It prompts for consumer ID, previous units, current units, and consumer type, then prints the calculated energy charges.

```

37 #MY TASK-2 is find code from Task-1 and extend it to find energy charges that was based on domestic,commercial,industrial consumers in optimis
38 def calculate_energy_charges(previous_units, current_units, consumer_type):
39     units_consumed = current_units - previous_units
40     if consumer_type.lower() == 'domestic':
41         if units_consumed <= 100:
42             charges = units_consumed * 1.2
43         elif units_consumed <= 300:
44             charges = (100 * 1.2) + (units_consumed - 100) * 2.0
45         else:
46             charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
47     elif consumer_type.lower() == 'commercial':
48         if units_consumed <= 100:
49             charges = units_consumed * 2.5
50         elif units_consumed <= 300:
51             charges = (100 * 2.5) + (units_consumed - 100) * 4.0
52         else:
53             charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
54     elif consumer_type.lower() == 'industrial':
55         if units_consumed <= 100:
56             charges = units_consumed * 3.0
57         elif units_consumed <= 300:
58             charges = (100 * 3.0) + (units_consumed - 100) * 5.0
59         else:
60             charges = (100 * 3.0) + (200 * 5.0) + (units_consumed - 300) * 7.5
61
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING> & C:\Users\NIRNAYA\AppData\Local\Microsoft\WindowsApps\python3.13.exe c:/Users/NIRNAYA/Desktop/AI_ASSISTANT_CODING/lab-3.3.py
Enter Consumer ID: 02
Enter Previous Units: 300
Enter Current Units: 550
Enter Consumer Type (domestic/commercial/industrial): domestic
Consumer ID: 02
Units Consumed: 250
Energy Charges: ₹420.00

```

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:
        print("Unknown consumer type")
    return charges

```

```

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 :

The screenshot shows a code editor interface with several tabs open. The active tab is 'lab-3.3.py'. The code in the editor is as follows:

```
my Task-3 is to find billing logic must be reusable for multiple consumers and that uses user defined functions to calculate energy charges, fixed charges and total bill.  
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
```

Below the code editor, the terminal window shows the execution of the script and its output:

```
PS C:\Users\NIRNAYA\OneDrive\Desktop\AI_ASSISTANT_CODING> & c:/Users/NIRNAYA/AppData/Local/Microsoft/WindowsApps/python3.13.exe c:/Users/NIRNAYA/OneDrive/Desktop/AI_ASSISTANT_CODING/lab-3.py  
Enter Previous ID: 0  
Enter Previous Units: 200  
Enter Current Units: 400  
Enter Consumer Type (domestic/commercial/industrial): domestic  
Consumer ID: 03  
Units consumed: 200  
Energy Charges: ₹200.00  
Fixed Charges: ₹50.00  
Total Bill: ₹350.00
```

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 :

```

File Edit Selection View Go Run Terminal Help < > AI ASSISTANT_CODING
lab-3.3.py lab-3.4.py lab-5.4.py lab-4.3.py lab1.py lab2.py
147 # my task - 4 is generate electricity bill including multiple additional charges like fixed charges,customer charges,percentage of electricity
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
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS QUERY RESULTS AZURE
Enter Consumer Type (domestic/commercial/industrial): industrial
Enter Fixed Charge 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 Charge: ₹1000.00
Customer Charges: ₹500.00
Electricity Duty: ₹5000.00
Total Bill: ₹6850.00
----- 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 :

```

def 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 :

The screenshot shows a Visual Studio Code (VS Code) interface with the following details:

- File Explorer:** Shows files in the "AI ASSISTANT_CODING" folder: lab-3.3.py, lab-3.4.py, lab-5.4.py, lab-4.3.py, lab1.py, and lab2.py.
- Code Editor:** Displays Python code for generating an electricity bill. The code includes comments explaining the task, imports, and a main function. It calculates various charges based on consumer ID, units consumed, and other parameters, and prints the total bill amount.
- Terminal:** Shows the output of the code execution, displaying the bill details and the total bill amount.
- Status Bar:** Shows the file path (C:\Users\INTRINXIA\OneDrive\Desktop\AI ASSISTANT_CODING), the number of lines (Line 268, Col 1), and the programming language (Python 3.13).

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.