

## ASSIGNMENT – 3.3

2303A51525

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
    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 :

The screenshot shows a code editor interface with a dark theme. The left sidebar lists files: lab-3.3.py (selected), lab-3.4.py, lab-4.3.py, lab-3.3.py, lab-5.4.py, lab-6.3.py, and lab-6.4.py. The main area displays Python code for calculating electricity bills based on consumption units and customer type (residential or commercial). The terminal at the bottom shows the execution of the script and its output.

```
1 #Task-1
2 #I want application for TGNPCL electricity bill payment system generation that must be collect accurate consumer data
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:
```

### 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): ") #
```

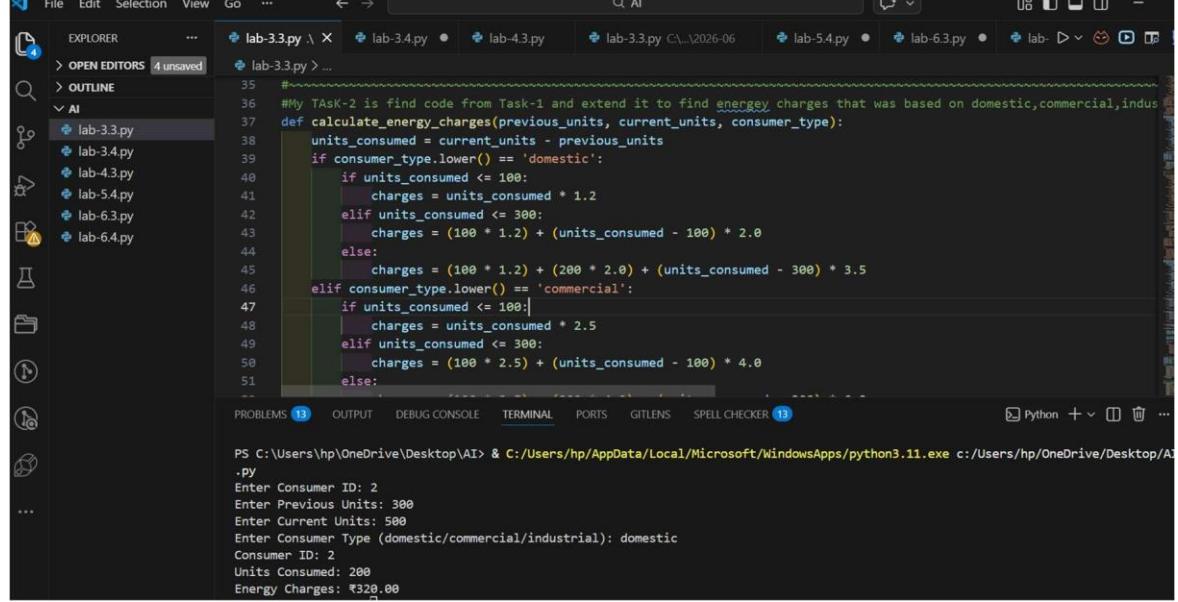
Calculating energy charges

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 :



The screenshot shows the Visual Studio Code interface. The left sidebar has an 'EXPLORER' view with several Python files listed: lab-3.3.py (selected), lab-3.4.py, lab-4.3.py, lab-3.3.py (C:\...\2026-06), lab-5.4.py, lab-6.3.py, and lab-6.4.py. The main area is the code editor with the following content:

```
35 #~  
36 #My Task-2 is find code from Task-1 and extend it to find energy charges that was based on domestic,commercial,indus  
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:
```

The terminal at the bottom shows the execution of the script:

```
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI  
.py  
Enter Consumer ID: 2  
Enter Previous Units: 300  
Enter Current Units: 500  
Enter Consumer Type (domestic/commercial/industrial): domestic  
Consumer ID: 2  
Units Consumed: 200  
Energy Charges: ₹320.00
```

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 ..... 3
            charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3   elif
        consumer_type.lower() == 'commercial': ..... 3
            if units_consumed <= 100:      charges = units_consumed * 2 ..... 3
            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 ..... 3
                elif units_consumed <= 300:      charges = (100 * 1.2) + (units_consumed - 100) * 2.0
                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): ")

try:
    bill_details = generate_bill(consumer_id, previous_units, current_units, consumer_type)
    print(f"Consumer ID: {bill_details['Consumer ID']}")  # Generating bill
    print(f"Units Consumed: {bill_details['Units Consumed']}")
    print(f"Energy Charges: {bill_details['Energy Charges']}")
    print(f"Fixed Charges: {bill_details['Fixed Charges']}")
    print(f"Total Bill: {bill_details['Total Bill']}")

```

```

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 :

```

File Edit Selection View Go ...
EXPLORER OPEN EDITORS 4 unsaved OUTLINE AI
lab-3.3.py > ...
79     #my Task-3 is to find billing logic must be reusable for multiple consumers and that uses user defined functions to c
80     def calculate_energy_charges(previous_units, current_units, consumer_type):
81         units_consumed = current_units - previous_units
82         if consumer_type.lower() == 'domestic':
83             if units_consumed <= 100:
84                 charges = units_consumed * 1.2
85             elif units_consumed <= 300:
86                 charges = (100 * 1.2) + (units_consumed - 100) * 2.0
87             else:
88                 charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5
89         elif consumer_type.lower() == 'commercial':
90             if units_consumed <= 100:
91                 charges = units_consumed * 2.5
92             elif units_consumed <= 300:
93                 charges = (100 * 2.5) + (units_consumed - 100) * 4.0
94             else:
95                 charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0
PROBLEMS 13 OUTPUT DEBUG CONSOLE TERMINAL PORTS GITLENS SPELL CHECKER 13 Python + ...
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-3.3
.py
Enter Consumer ID: 03
Enter Previous Units: 200
Enter Current Units: 600
Enter Consumer Type (domestic/commercial/industrial): domestic
Consumer ID: 03
Units Consumed: 400
Energy Charges: ₹870.00
Fixed Charges: ₹50.00
Total Bill: ₹920.00

```

### 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
```

The screenshot shows a code editor interface with multiple tabs open. The active tab is 'lab-33.py'. The code in the editor is a function named 'calculate\_energy\_charges' that takes three parameters: 'previous\_units', 'current\_units', and 'consumer\_type'. The function calculates various charges based on consumer type (domestic, commercial, industrial) and previous/current units consumed. The output tab shows the execution of the code with user input for previous units (200), current units (234), consumer type (industrial), fixed charges (2000), customer charges (300), and electricity duty percentage (4%). The total bill calculated is ₹60729.00.

```
146 #my task -4 is generate electricity bill including multiple additional charges like fixed charges,customer charges,per unit charges etc  
147 def calculate_energy_charges(previous_units, current_units, consumer_type):  
148     units_consumed = current_units - previous_units  
149     if consumer_type.lower() == 'domestic':  
150         if units_consumed <= 100:  
151             charges = units_consumed * 1.2  
152         elif units_consumed <= 300:  
153             charges = (100 * 1.2) + (units_consumed - 100) * 2.0  
154         else:  
155             charges = (100 * 1.2) + (200 * 2.0) + (units_consumed - 300) * 3.5  
156     elif consumer_type.lower() == 'commercial':  
157         if units_consumed <= 100:  
158             charges = units_consumed * 2.5  
159         elif units_consumed <= 300:  
160             charges = (100 * 2.5) + (units_consumed - 100) * 4.0  
161         else:  
162             charges = (100 * 2.5) + (200 * 4.0) + (units_consumed - 300) * 6.0  
163  
.py  
Enter Previous Units: 200  
Enter Current Units: 234  
Enter Consumer Type (domestic/commercial/industrial): industrial  
Enter Fixed Charges for Industrial: 2000  
Enter Customer Charges for Industrial: 300  
Enter Electricity Duty Percentage: 490  
Consumer ID: 1  
Units Consumed: 34  
Energy Charges: ₹119.00  
Fixed Charges: ₹2000.00  
Customer Charges: ₹300.00  
Electricity Duty: ₹58310.00  
Total Bill: ₹60729.00
```

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 :

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

```
238 #Task-5 is generate final bill of electricity including all charges with proper formatting and display in python lang
239 def display_bill(bill_details):
240     print("----- Electricity Bill -----")
241     print(f"Consumer ID : {bill_details['Consumer ID']}")
242     print(f"Units Consumed : {bill_details['Units Consumed']} units")
243     print(f"Energy Charges : ₹{bill_details['Energy Charges']:.2f}")
244     print(f"Fixed Charges : ₹{bill_details['Fixed Charges']:.2f}")
245     print(f"Customer Charges : ₹{bill_details['Customer Charges']:.2f}")
246     print(f"Electricity Duty : ₹{bill_details['Electricity Duty']:.2f}")
247     print("-----")
248     print(f"Total Bill Amount : ₹{bill_details['Total Bill']:.2f}")
249     print("-----")
250 # Example usage
251 consumer_id = "C12345"
252 previous_units = 500
253 current_units = 750
254 consumer_type = "domestic"
```

The terminal below the editor shows the output of running the script:

```
PS C:\Users\hp\OneDrive\Desktop\AI> & C:/Users/hp/AppData/Local/Microsoft/WindowsApps/python3.11.exe c:/Users/hp/OneDrive/Desktop/AI/lab-3.3.py
----- 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 :

- 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.