

ASSIGNMENT – 3.3

2303A51197

Batch-10

Task-1

Prompt: Create a Python program for TGNPDCL electricity billing that takes previous units, current units, and customer type, then calculates the bill. 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 :

```

lab_3.3.py > ...
1  #Create a Python program for TGNPDCL electricity billing that takes previous units, current units, and customer type, then calculates the bill amount.
2  class ElectricityBill:
3      def __init__(self, previous_units, current_units, customer_type):
4          self.previous_units = previous_units
5          self.current_units = current_units
6          self.customer_type = customer_type
7
8      def calculate_bill(self):
9          units_consumed = self.current_units - self.previous_units
10         if self.customer_type == "Residential":
11             if units_consumed <= 100:
12                 bill_amount = units_consumed * 0.5
13             elif units_consumed <= 200:
14                 bill_amount = (100 * 0.5) + ((units_consumed - 100) * 0.75)
15             else:
16                 bill_amount = (100 * 0.5) + (100 * 0.75) + ((units_consumed - 200) * 1.0)
17         elif self.customer_type == "Commercial":
18             if units_consumed <= 100:
19                 bill_amount = units_consumed * 1.0
20             elif units_consumed <= 200:
21                 bill_amount = (100 * 1.0) + ((units_consumed - 100) * 1.5)
22             else:
23                 bill_amount = (100 * 1.0) + (100 * 1.5) + ((units_consumed - 200) * 2.0)
24         else:
25             raise ValueError("Invalid customer type")
26         return bill_amount
27     def main():
28         previous_units = int(input("Enter previous units: "))

```

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: Improve the previous program to calculate energy charges for domestic, commercial, and industrial users using better condition statements.

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 :

```

print(f" The electricity bill is: {bill_amount} ")
#Improve the previous program to calculate energy charges for domestic, commercial, and industrial users using better condition statements.
class Electricity:
    def __init__(self, previous_units, current_units, customer_type):
        self.previous_units = previous_units
        self.current_units = current_units
        self.customer_type = customer_type
    def calculate_bill(self):
        units_consumed = self.current_units - self.previous_units
        if self.customer_type == "residential":
            if units_consumed <= 100:
                bill = units_consumed * 0.5
            elif units_consumed <= 200:
                bill = (100 * 0.5) + ((units_consumed - 100) * 0.75)
            else:
                bill = (100 * 0.5) + (100 * 0.75) + ((units_consumed - 200) * 1)
        elif self.customer_type == "commercial":
            if units_consumed <= 100:
                bill = units_consumed * 0.75
            elif units_consumed <= 200:
                bill = (100 * 0.75) + ((units_consumed - 100) * 1)
            else:
                bill = (100 * 0.75) + (100 * 1) + ((units_consumed - 200) * 1.25)
        elif self.customer_type == "industrial":
            if units_consumed <= 100:
                bill = units_consumed * 1
            elif units_consumed <= 200:
                bill = (100 * 1) + ((units_consumed - 100) * 1.25)
            else:
                bill = (100 * 1) + (100 * 1.25) + ((units_consumed - 200) * 1.5)

```

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: Write reusable Python functions to calculate energy charges and fixed charges for different consumers and return the bill details.

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:
        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 :

```
72
73     #Write reusable Python functions to calculate energy charges and fixed charges for different consumers and return the bill details.
74     def calculate_energy_charges(units_consumed, customer_type):
75         if customer_type == "residential":
76             if units_consumed <= 100:
77                 return units_consumed * 0.5
78             elif units_consumed <= 200:
79                 return (100 * 0.5) + ((units_consumed - 100) * 0.75)
80             else:
81                 return (100 * 0.5) + (100 * 0.75) + ((units_consumed - 200) * 1)
82         elif customer_type == "commercial":
83             if units_consumed <= 100:
84                 return units_consumed * 0.75
85             elif units_consumed <= 200:
86                 return (100 * 0.75) + ((units_consumed - 100) * 1)
87             else:
88                 return (100 * 0.75) + (100 * 1) + ((units_consumed - 200) * 1.25)
89         elif customer_type == "industrial":
90             if units_consumed <= 100:
91                 return units_consumed * 1
92             elif units_consumed <= 200:
93                 return (100 * 1) + ((units_consumed - 100) * 1.25)
94             else:
95                 return (100 * 1) + (100 * 1.25) + ((units_consumed - 200) * 1.5)
96         else:
97             return "Invalid customer type"
98     def calculate_fixed_charges(customer_type):
99         if customer_type == "residential":
100             return 50
101        elif customer_type == "commercial":
102            return 100
103        elif customer_type == "industrial":
104            return 150
105        else:
106            return "Invalid customer type"
107    def calculate_total_bill(previous_units, current_units, customer_type):
108        units_consumed = current_units - previous_units
109        energy_charges = calculate_energy_charges(units_consumed, customer_type)
110        fixed_charges = calculate_fixed_charges(customer_type)
111        if isinstance(energy_charges, str) or isinstance(fixed_charges, str):
112            return "Invalid customer type"
113        total_bill = energy_charges + fixed_charges
```

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 ... OPEN EDITORS 5 unsaved EXPLORER lab-3.3.py > ... OUTLINE AI lab-3.3.py lab-3.4.py lab-4.3.py lab-3.3.py > ... 146 #my task -4 is generate electricity bill including multiple additional charges like fixed charges,customer charges,pe 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     return charges 164 165 .py Enter Previous Units: 200 Enter Current Units: 234 Enter Consumer Type (domestic/commercial/industrial): industrial Enter Fixed Charges for Industrial: 300 Enter Customer Charges for Industrial: 400 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("-----\n")
250 # Example usage
251 consumer_id = "C12345"
252 previous_units = 500
253 current_units = 750
254 consumer_type = "domestic"
```

Below the code, the terminal output shows the execution of the script with sample data:

```
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.