ASSIGNMENT-3.3

2303A51337

BATCH-10

➢ TASK-1:

PROMPT:

PYTHON code for reading consumer details and generate a python program it should read previous units and current units and type of customer (domestic or commercial) and calculate the units consumed program without function

Reading consumer details and calculating units consumed

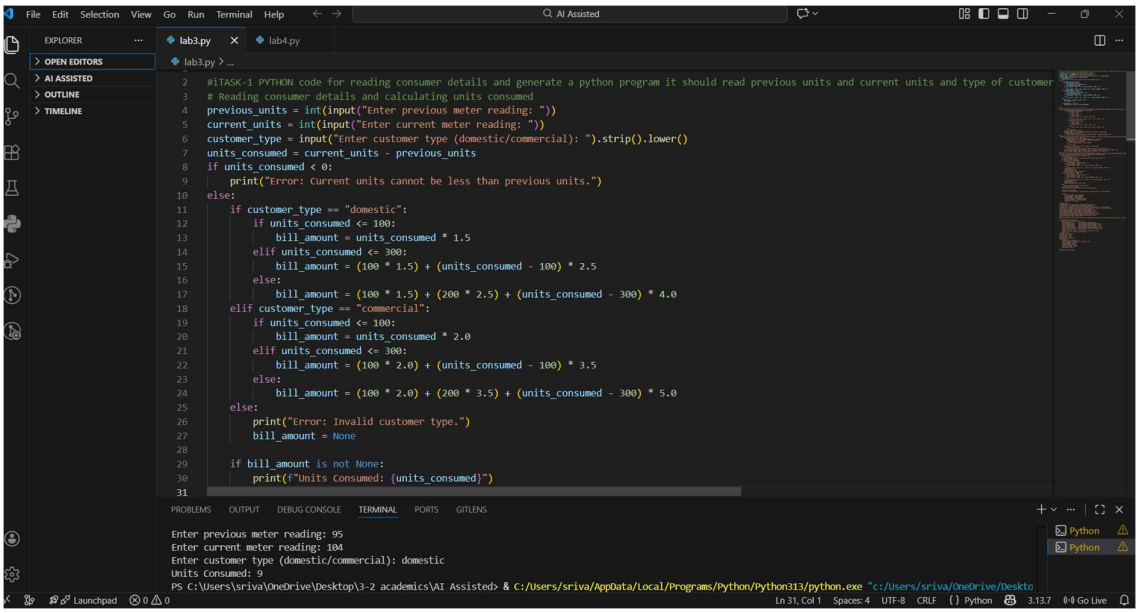
CODE:

```
previous_units = int(input("Enter previous meter reading: "))
current_units = int(input("Enter current meter reading: "))
customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
units_consumed = current_units - previous_units
if units_consumed < 0:
  print("Error: Current units cannot be less than previous units.")
else:
  if customer_type == "domestic":
    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 == "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:
    print("Error: Invalid customer type.")
    bill_amount = None

if bill_amount is not None:
    print(f"Units Consumed: {units_consumed}")
```

OUTPUT:



EXPLANATION:

- This task focuses on collecting correct consumer details required for electricity billing.
- The program reads previous units, current units, and consumer type from the user.
- Units consumed are calculated using meter readings, which is the base for billing.
- This step ensures accurate input handling for further calculations.

❖ TASK-2

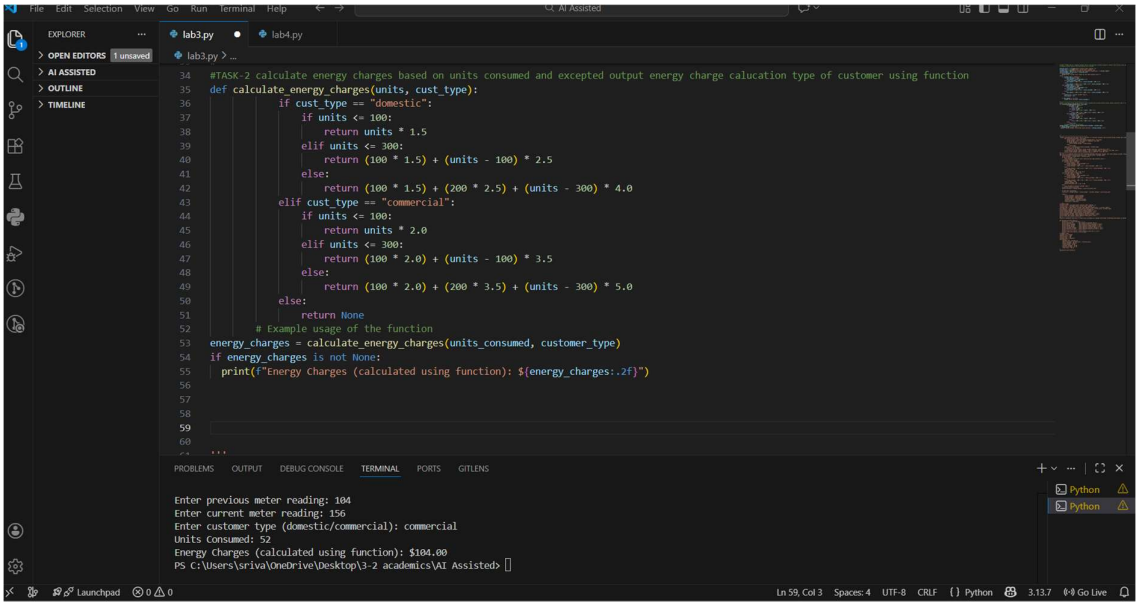
PROMPT:

calculate energy charges based on units consumed and excepted output energy charge calucation type of customer using function.

CODE:

```
def calculate_energy_charges(units, cust_type):
        if cust_type == "domestic":
          if units <= 100:
            return units * 1.5
          elif units <= 300:
            return (100 * 1.5) + (units - 100) * 2.5
          else:
            return (100 * 1.5) + (200 * 2.5) + (units - 300) * 4.0
        elif cust_type == "commercial":
          if units <= 100:
            return units * 2.0
          elif units <= 300:
            return (100 * 2.0) + (units - 100) * 3.5
          else:
            return (100 * 2.0) + (200 * 3.5) + (units - 300) * 5.0
        else:
          return None
    # Example usage of the function
energy_charges = calculate_energy_charges(units_consumed, customer_type)
if energy_charges is not None:
  print(f"Energy Charges (calculated using function): ${energy_charges:.2f}")
```

OUTPUT:



EXPLANATION:

In this task, energy charges are calculated based on units consumed and consumer type.

Conditional logic is used to apply different tariff rates for domestic, commercial, and industrial users.

❖ TASK-3:

PROMPT:

calculate billing logic must be resusable for multiple consumers and calculate energy charges and fixed charges return calculated values

CODE:

```
def calculate_bill(units, cust_type):
    energy_charges = calculate_energy_charges(units, cust_type)
    fixed_charges = 50 if cust_type == "domestic" else 100
    if energy_charges is not None:
      return energy_charges + fixed_charges
    else:
      return None
```

total_bill = calculate_bill(units_consumed, customer_type)

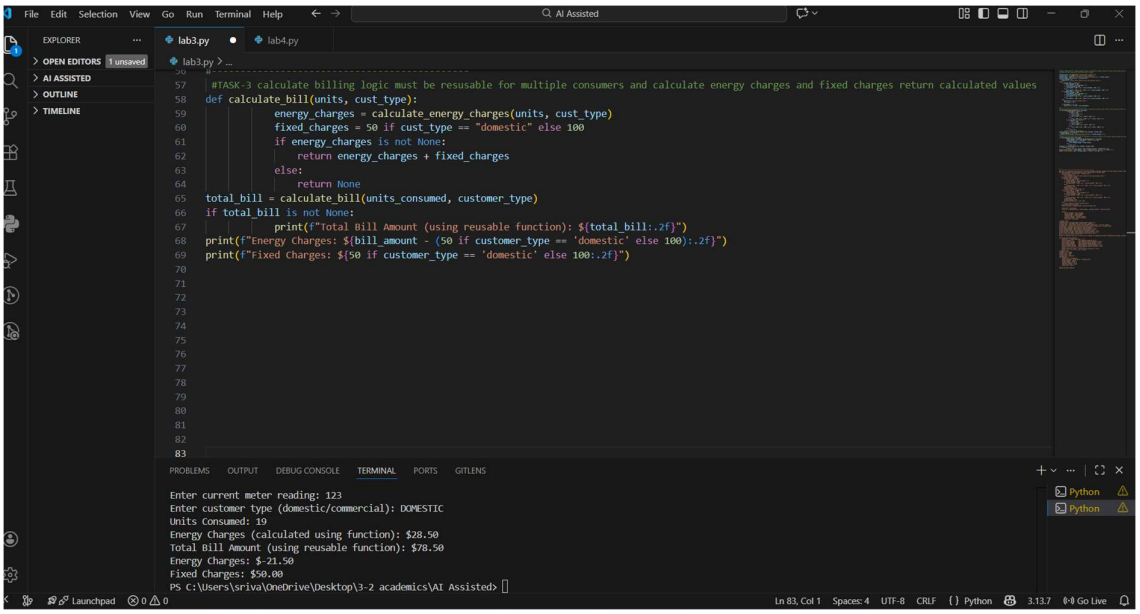
if total_bill is not None:

print(f"Total Bill Amount (using reusable function): ${total_bill:.2f}")

print(f"Energy Charges: ${bill_amount - (50 if customer_type == 'domestic' else 100):.2f}")

print(f"Fixed Charges: ${50 if customer_type == 'domestic' else 100:.2f}")

OUTPUT:



EXPLANATION:

Introduces modular programming using user-defined methods.

Separate methods are used to calculate energy charges and fixed charges.

This makes the program reusable and easier to maintain.

Modular design improves code structure and readability.

❖ TASK-4:

PROMPT:

Generate electricity bill including multiple additional charges like fixed charges,customer charges,percentage of electricity duty,duty calulation by improving accuracY

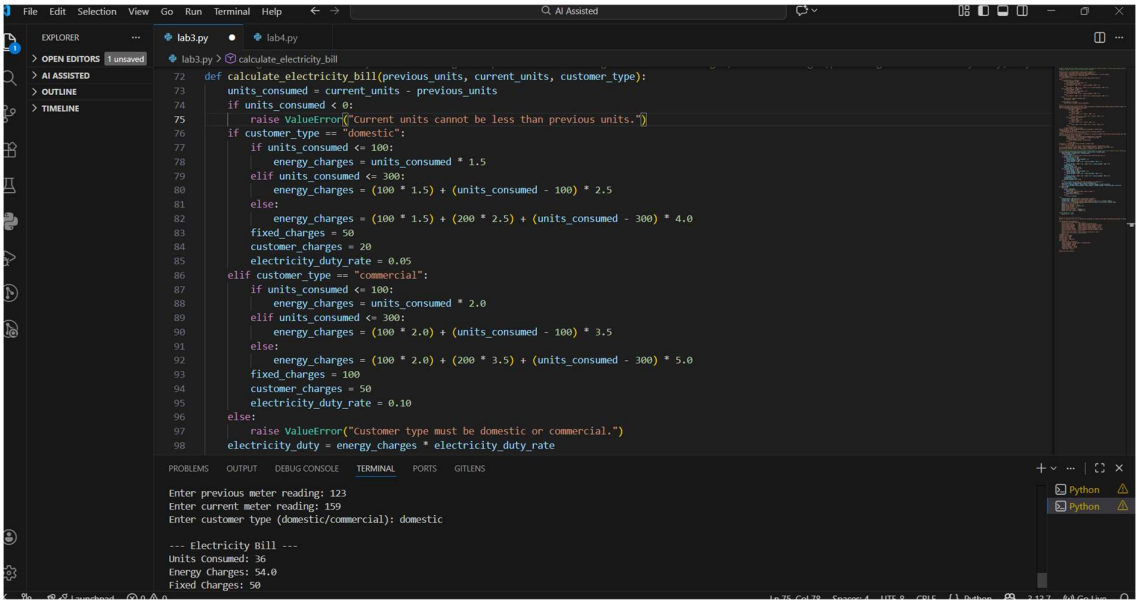
CODE:

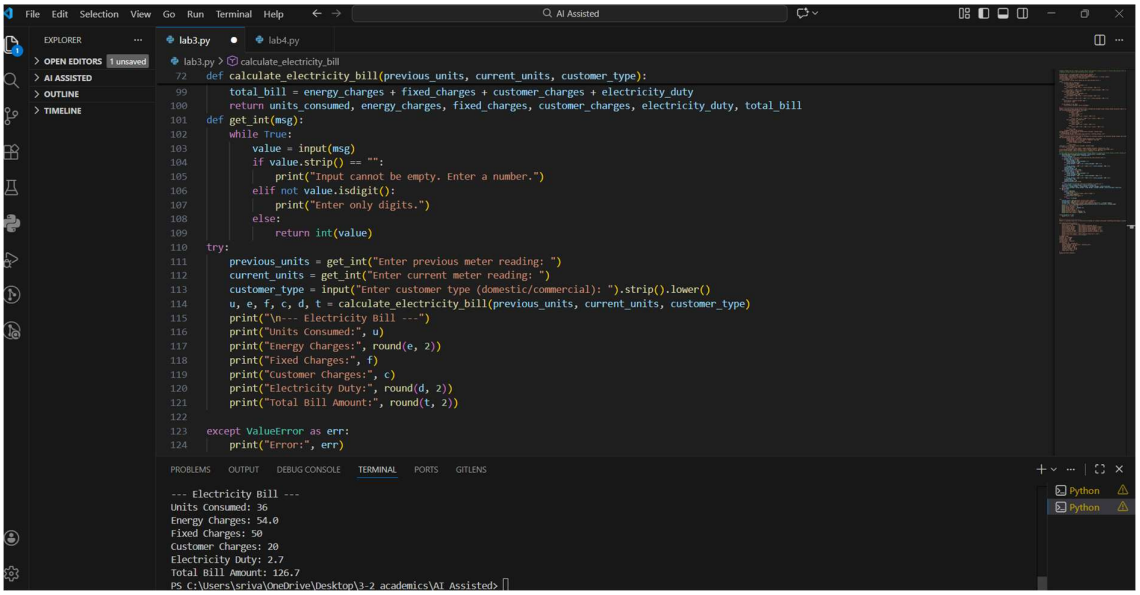
```
def calculate_electricity_bill(previous_units, current_units, customer_type):
    units_consumed = current_units - previous_units
    if units_consumed < 0:
        raise ValueError("Current units cannot be less than previous units.")
    if customer_type == "domestic":
        if units_consumed <= 100:
            energy_charges = units_consumed * 1.5
        elif units_consumed <= 300:
            energy_charges = (100 * 1.5) + (units_consumed - 100) * 2.5
        else:
            energy_charges = (100 * 1.5) + (200 * 2.5) + (units_consumed - 300) * 4.0
        fixed_charges = 50
        customer_charges = 20
        electricity_duty_rate = 0.05
    elif customer_type == "commercial":
        if units_consumed <= 100:
            energy_charges = units_consumed * 2.0
        elif units_consumed <= 300:
            energy_charges = (100 * 2.0) + (units_consumed - 100) * 3.5
        else:
            energy_charges = (100 * 2.0) + (200 * 3.5) + (units_consumed - 300) * 5.0
        fixed_charges = 100
        customer_charges = 50
        electricity_duty_rate = 0.10
    else:
        raise ValueError("Customer type must be domestic or commercial.")
    electricity_duty = energy_charges * electricity_duty_rate
```

```
    total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
    return units_consumed, energy_charges, fixed_charges, customer_charges,
electricity_duty, total_bill
def get_int(msg):
    while True:
        value = input(msg)
        if value.strip() == "":
            print("Input cannot be empty. Enter a number.")
        elif not value.isdigit():
            print("Enter only digits.")
        else:
            return int(value)
try:
    previous_units = get_int("Enter previous meter reading: ")
    current_units = get_int("Enter current meter reading: ")
    customer_type = input("Enter customer type (domestic/commercial): ").strip().lower()
    u, e, f, c, d, t = calculate_electricity_bill(previous_units, current_units, customer_type)
    print("\n--- Electricity Bill ---")
    print("Units Consumed:", u)
    print("Energy Charges:", round(e, 2))
    print("Fixed Charges:", f)
    print("Customer Charges:", c)
    print("Electricity Duty:", round(d, 2))
    print("Total Bill Amount:", round(t, 2))

except ValueError as err:
    print("Error:", err)
```

OUTPUT:





EXPLANATION:

This task extends billing by adding additional charges like fixed charges, customer charges, and electricity duty.

Electricity duty is calculated as a percentage of energy charges.

Printing individual charges helps verify calculation accuracy.

This step makes the bill more realistic and detailed.

❖ TASK-5:

PROMPT:

Generate final bill of electricity including all charges with proper formatting and display in python .

CODE:

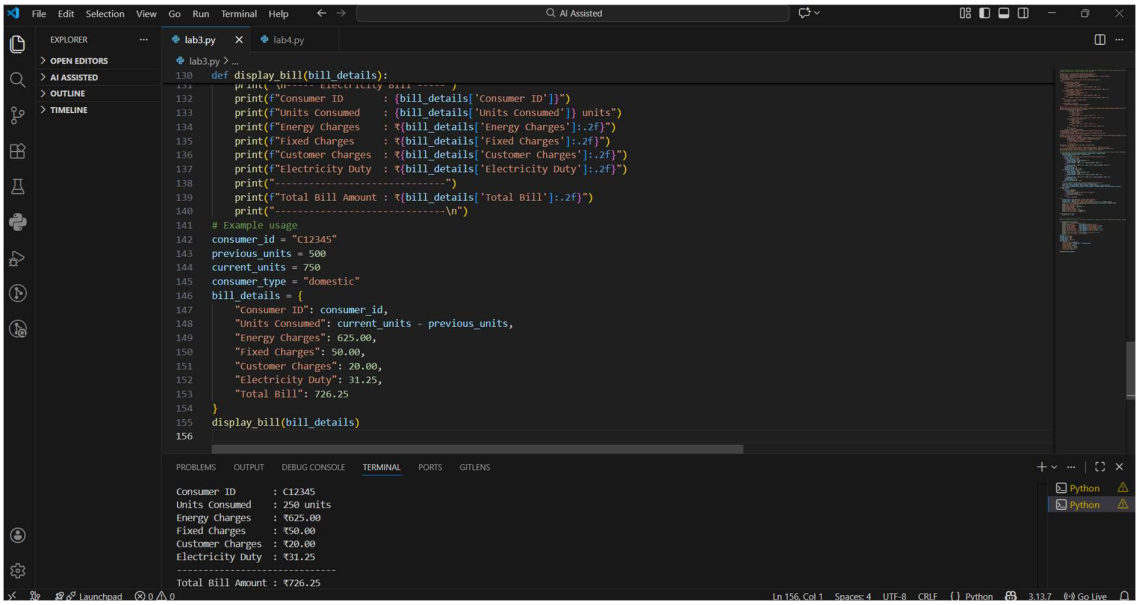
```
def display_bill(bill_details):
    print("\n----- Electricity Bill -----")
    print(f"Consumer ID      : {bill_details['Consumer ID']}")
    print(f"Units Consumed    : {bill_details['Units Consumed']} units")
    print(f"Energy Charges    : ₹{bill_details['Energy Charges']:.2f}")
    print(f"Fixed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



EXPLANATION:

final electricity bill by combining all charge components.

The total bill amount is calculated by adding EC, FC, CC, and ED.

The output is displayed in a neat, bill-like format for clarity.

This task demonstrates a complete, real-world electricity billing application