Name: M.sivateja

Hall Ticket: 2303A52145

Batch: 41

Task 1: AI-Generated Logic for Reading Consumer Details Scenario

Prompt:

Write a Python program (no functions) to read electricity_data.csv from Google Colab, validate that current units are greater than or equal to previous units, calculate units consumed, and display consumer ID, customer type, previous units, current units, and units consumed in a clear format.

Updated Prompt:

Enhance the existing Python program by adding logic to detect unusual conditions such as high electricity usage or invalid readings, without changing the current output format. Display a simple informational flag a er each consumer's details while keeping all original output lines unchanged.

Code:

```python
import math

def add(numbers):
    return sum(numbers)

def subtract(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        result -= num
    return result

def multiply(numbers):
    result = 1
    for num in numbers:
        result *= num
    return result

def divide(numbers):
    result = numbers[0]
    for num in numbers[1:]:
        if num == 0:
            return "Error: Division by zero"
        result /= num
    return result

def square_root(number):
    if number < 0:
        return "Error: Cannot calculate square root of a negative number"
    return math.sqrt(number)

def check_anomaly(result):
    """Logic to detect unusual conditions like high usage or invalid readings."""
    if isinstance(result, str):
        return "[FLAG: INVALID READING]"
    if result > 1000:
        return "[FLAG: HIGH USAGE DETECTED]"
```

```
    return "[OK]"

def main():
    print("--- Simple Python Calculator ---")
    print("1. Addition")
    print("2. Subtraction")
    print("3. Multiplication")
    print("4. Division")
    print("5. Square Root")

    choice = input("\nSelect an operation (1-5): ")

    if choice in ['1', '2', '3', '4']:
        count = int(input("How many numbers do you want to enter? "))
        nums = []
        for i in range(count):
            val = float(input(f"Enter number {i+1}: "))
            nums.append(val)

        res = 0
        if choice == '1':
            res = add(nums)
        elif choice == '2':
            res = subtract(nums)
        elif choice == '3':
            res = multiply(nums)
        elif choice == '4':
            res = divide(nums)

        print(f"Result: {res} {check_anomaly(res)}")

    elif choice == '5':
        num = float(input("Enter a number for square root: "))
        res = square_root(num)
        print(f"Result: {res} {check_anomaly(res)}")
    else:
        print("Invalid selection. Please run the program again.")
```

```
if __name__ == '__main__':
    main()
```

Output:

```
--- Simple Python Calculator ---
1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Square Root

Select an operation (1-5): 4
How many numbers do you want to enter? 2
Enter number 1: 3
Enter number 2: 2
Result: 1.5 [OK]
```

TASK-2:

PROMPT:

Extend the electricity billing code to calculate energy charges based on customer type using if-else conditions for Domestic, Commercial, and Industrial consumers.”

Code:

```
def calculate_electricity_bill(units, customer_type):
    customer_type = customer_type.strip().lower()
    rate = 0.0

    # Determine rate using if-else conditions
    if customer_type == "domestic":
        rate = 0.50
    elif customer_type == "commercial":
        rate = 0.75
    elif customer_type == "industrial":
        rate = 1.20
    else:
        return None # Invalid customer type

    return units * rate

def main():
    print("--- Electricity Bill Calculator ---")
    try:
        c_type = input("Enter Customer Type (Domestic, Commercial, Industrial): ")
        usage = float(input("Enter total units consumed: "))

        if usage < 0:
            print("Error: Units consumed cannot be negative.")
        else:
            total_charge = calculate_electricity_bill(usage, c_type)

            if total_charge is not None:
                print(f"\nSummary for {c_type.capitalize()} Customer:")
                print(f"Units Consumed: {usage}")
                print(f"Total Energy Charges: ${total_charge:.2f}")
            else:
                print("Error: Invalid customer type entered.")
    except ValueError:
        print("Error: Please enter a numeric value for units.")

if __name__ == '__main__':
    main()
```

OUTPUT:

```
--- Electricity Bill Calculator ---
Enter Customer Type (Domestic, Commercial, Industrial): domestic
Enter total units consumed: 4

Summary for Domestic Customer:
Units Consumed: 4.0
Total Energy Charges: $2.00
```

4.Prompt Improvement with AI:

“Simplify energy charge calculation logic”

“Optimize conditional statements for better readability”

```python
def calculate_bill(units, customer_type):
    # Mapping customer types to their respective rates
    rates = {
        "domestic": 0.50,
        "commercial": 0.75,
        "industrial": 1.20
    }

    # Lookup the rate using the dictionary
    rate = rates.get(customer_type.strip().lower())

    if rate is None:
        return "Error: Invalid customer type."

    return units * rate

def main():
    print("--- Optimized Electricity Bill Calculator ---")
    try:
        c_type = input("Enter Customer Type (Domestic, Commercial, Industrial): ")
        usage = float(input("Enter units consumed: "))

        if usage < 0:
            print("Error: Units consumed cannot be negative.")
        else:
            result = calculate_bill(usage, c_type)
            if isinstance(result, str):
                print(result)
            else:
                print(f"\nSummary for {c_type.capitalize()}:")
                print(f"Usage: {usage} units")
                print(f"Total Charge: ${result:.2f}")
    except ValueError:
        print("Error: Please enter a valid number for units.")

if __name__ == '__main__':
    main()
```

5.Comparision bet

What I changed and why:

- Replaced the long if/elif chain with a simple dictionary lookup for rates to improve readability and make it easy to update or extend tariffs.
- Kept input parsing and validation compact and clear.
- Kept all logic inline (no functions) as requested.
- Next steps you might want: switch to slab-based rates (if tariffs are tiered), add taxes/fees, or validate/normalize customer-type variations further.

```python
# Simplified electricity billing calculator (no functions)
# - Uses a rates lookup dict instead of multiple if/elif branches
# - Clear, compact validation and output

prev_input = input("Enter previous units: ").strip()
curr_input = input("Enter current units: ").strip()
customer_type = input("Enter customer type (Domestic/Commercial/Industrial): ").strip()

# Parse numeric inputs
try:
    previous = float(prev_input)
    current = float(curr_input)
except ValueError:
    print("Error: Previous and current units must be numeric.")
    raise SystemExit(1)

# Validate readings
if previous < 0 or current < 0:
    print("Error: Readings must be non-negative.")
    raise SystemExit(1)
if current < previous:
    print("Error: Current reading cannot be less than previous reading.")
    raise SystemExit(1)

units_consumed = current - previous

# Rate lookup (flat rates). Change values here to update tariffs.
rates = {
    "domestic": 1.50,
    "d": 1.50,
    "commercial": 3.00,
    "c": 3.00,
    "industrial": 2.50,
    "i": 2.50,
```

```
key = customer_type.strip().lower()
rate = rates.get(key)
if rate is None:
    print("Error: Unknown customer type. Use Domestic, Commercial, or Industrial (or D/C/I).")
    raise SystemExit(1)

energy_charge = units_consumed * rate

# Nicely format units (integer when whole) and currency
units_display = int(units_consumed) if units_consumed.is_integer() else round(units_consumed, 2)

print("\n--- Bill Summary ---")
print(f"Customer type: {customer_type.strip().title()}")
print(f"Previous reading: {previous}")
print(f"Current reading:  {current}")
print(f"Units consumed:   {units_display}")
print(f"Rate per unit:    {rate:.2f}")
print(f"Energy charge:    {energy_charge:.2f}")
```

```
Enter previous units: 1000
Enter current units: 1250
Enter customer type (Domestic/Commercial/Industrial): Domestic

--- Bill Summary ---
Customer type: Domestic
Previous reading: 1000.0
Current reading:  1250.0
Units consumed:   250
Rate per unit:    1.50
Energy charge:    375.00
```

TASK-03:

PROMPT:

“Rewrite the electricity billing program using functions to calculate Energy Charges and Fixed Charges. Include comments explaining each function.”

```
def calculate_energy_charge(units, customer_type):
    """
    lates the variable energy charge based on consumption and customer type.

    = {
        domestic": 0.50,
        commercial": 0.75,
        "industrial": 1.20
    }
    rate = rates.get(customer_type.lower())
    return units * rate if rate is not None else None

def get_fixed_charge(customer_type):
    """
    Returns the fixed monthly service fee based on the customer type.
    """
    fixed_fees = {
        "domestic": 10.00,
        "commercial": 25.00,
        "industrial": 50.00
    }
    return fixed_fees.get(customer_type.lower())

def main():
    print("--- Electricity Billing System ---")
    c_type = input("Enter Customer Type (Domestic, Commercial, Industrial): ").strip().lower()

    try:
        usage = float(input("Enter units consumed: "))
```

```
24s
(Ctrl+Enter)
cuted since last change

d by MUSKU SIVATEJA
(1 minute ago)
d in 24.343s
            energy_charge = calculate_energy_charge(usage, c_type)
            fixed_charge = get_fixed_charge(c_type)

            if energy_charge is None:
                print("Error: Invalid customer type.")
            elif usage < 0:
                print("Error: Usage cannot be negative.")
            else:
                total_bill = energy_charge + fixed_charge
                print(f"\n--- Bill Summary ({c_type.capitalize()}) ---")
                print(f"Energy Charges: ${energy_charge:.2f}")
                print(f"Fixed Charges:  ${fixed_charge:.2f}")
                print(f"Total Amount:   ${total_bill:.2f}")

        except ValueError:
            print("Error: Please enter a valid numerical value for units.")

    if __name__ == '__main__':
        main()
```

Output:

```
...  --- Electricity Billing System ---
     Enter Customer Type (Domestic, Commercial, Industrial): industrial
     Enter units consumed: 60

     --- Bill Summary (Industrial) ---
     Energy Charges: $72.00
     Fixed Charges:  $50.00
     Total Amount:   $122.00
```

Task4:

```python
# Electricity billing program using functions
# - calculate_energy_charge(units, customer_type): computes energy charge from per-unit rates
# - calculate_fixed_charge(customer_type): returns fixed monthly/standing charge for customer type
# - calculate_customer_charge(customer_type): returns customer-specific charge (additional)
# - calculate_electricity_duty(energy_charge): computes duty as 10% of energy charge
# The main section reads inputs, validates them, calls the functions, and prints the bill.

def calculate_energy_charge(units, customer_type):
    """
    Calculate the energy (consumption) charge.
    Parameters:
    - units (float): number of units consumed
    - customer_type (str): customer category (Domestic, Commercial, Industrial, or short forms D/C/I)
    Returns:
    - tuple (energy_charge (float), rate_per_unit (float))
    Raises:
    - ValueError if customer_type is unrecognized
    Notes:
    - Uses flat per-unit rates. Modify the 'rates' dict to change tariffs.
    """
    rates = {
        "domestic": 1.50,
        "d": 1.50,
        "commercial": 3.00,
        "c": 3.00,
        "industrial": 2.50,
        "i": 2.50,
    }

    key = customer_type.strip().lower()
    rate = rates.get(key)
    if rate is None:
        raise ValueError("Unknown customer type for energy rate. Use Domestic, Commercial, or Industrial (or D

    energy_charge = units * rate
    return energy_charge, rate


def calculate_fixed_charge(customer_type):
    """
    Return the fixed (standing) charge for the billing period based on customer type.
    Parameters:
    - customer_type (str): customer category (Domestic, Commercial, Industrial, or short forms D/C/I)
    Returns:
    - fixed_charge (float)
Raises:
- ValueError if customer_type is unrecognized
Notes:
- Modify the 'fixed_charges' dict to change values.
"""
fixed_charges = {
    "domestic": 50.00,
    "d": 50.00,
    "commercial": 100.00,
    "c": 100.00,
    "industrial": 200.00,
    "i": 200.00,
}

key = customer_type.strip().lower()
fixed = fixed_charges.get(key)
if fixed is None:
    raise ValueError("Unknown customer type for fixed charge. Use Domestic, Commercial, or Industrial (or
return fixed
```

```python
def calculate_customer_charge(customer_type):
    """
    Return a customer charge which is an additional charge depending on customer type.
    Parameters:
    - customer_type (str): customer category (Domestic, Commercial, Industrial, or short forms D/C/I)
    Returns:
    - customer_charge (float)
    Raises:
    - ValueError if customer_type is unrecognized
    Notes:
    - These are example values to illustrate a per-customer-type surcharge.
    """
    customer_charges = {
        "domestic": 30.00,
        "d": 30.00,
        "commercial": 60.00,
        "c": 60.00,
        "industrial": 120.00,
        "i": 120.00,
    }

    key = customer_type.strip().lower()
    cust_charge = customer_charges.get(key)
    if cust_charge is None:
        raise ValueError("Unknown customer type for customer charge. Use Domestic, Commercial, or Industrial (
    return cust_charge
```

```python
def calculate_electricity_duty(energy_charge):
    """
    Calculate electricity duty as 10% of the energy charge.
    Parameters:
    - energy_charge (float)
    Returns:
    - duty (float)
    """
    duty_rate = 0.10  # 10%
    return energy_charge * duty_rate
```

```python
    # Read inputs
    prev_input = input("Enter previous units: ").strip()
    curr_input = input("Enter current units: ").strip()
    customer_type = input("Enter customer type (Domestic/Commercial/Industrial): ").strip()

    # Parse numeric inputs
    try:
        previous = float(prev_input)
        current = float(curr_input)
    except ValueError:
        print("Error: Previous and current units must be numeric.")
        raise SystemExit(1)

    # Validate readings
    if previous < 0 or current < 0:
        print("Error: Readings must be non-negative.")
        raise SystemExit(1)
    if current < previous:
        print("Error: Current reading cannot be less than previous reading.")
        raise SystemExit(1)
```

```
# Compute units consumed
units_consumed = current - previous

# Use functions to compute charges
try:
    energy_charge, rate_per_unit = calculate_energy_charge(units_consumed, customer_type)
    fixed_charge = calculate_fixed_charge(customer_type)
    customer_charge = calculate_customer_charge(customer_type)
    electricity_duty = calculate_electricity_duty(energy_charge)
except ValueError as e:
    print(f"Error: {e}")
    raise SystemExit(1)

# Total bill
total_amount = energy_charge + fixed_charge + customer_charge + electricity_duty

# Nicely format units (integer when whole) and currency
units_display = int(units_consumed) if units_consumed.is_integer() else round(units_consumed, 2)

print("\n--- Bill Summary ---")
print(f"Customer type:      {customer_type.strip().title()}")
print(f"Previous reading:   {previous}")
print(f"Current reading:    {current}")
print(f"Units consumed:     {units_display}")
print(f"Rate per unit:      {rate_per_unit:.2f}")
print(f"Energy charge:      {energy_charge:.2f}")
print(f"Fixed charge:       {fixed_charge:.2f}")
print(f"Customer charge:    {customer_charge:.2f}")
print(f"Electricity duty (10%): {electricity_duty:.2f}")
print(f"Total amount due:   {total_amount:.2f}")
```

```
print(f"Total amount due:   {total_amount:.2f}")
```

OUTPUT:

```
Enter previous units: 1000
Enter current units: 1250
Enter customer type (Domestic/Commercial/Industrial): Domestic

--- Bill Summary ---
Customer type:      Domestic
Previous reading:   1000.0
Current reading:    1250.0
Units consumed:     250
Rate per unit:      1.50
Energy charge:      375.00
Fixed charge:       50.00
Customer charge:    30.00
Electricity duty (10%): 37.50
Total amount due:   492.50
```

Task-05

Prompt: Complete the electricity billing system by calculating and printing a detailed bill including EC, FC, CC, ED, and total.”

Comprehensive Electricity Billing System

This program calculates a detailed bill including:

- **EC (Energy Charge)**: Based on unit consumption.
- **FC (Fixed Charge)**: A flat rate per customer type.
- **CC (Customer Charge)**: Operational service fee.
- **ED (Electricity Duty)**: Tax (5% of EC).
- **Total**: Sum of all components.

```
def get_bill_components(units, customer_type):
    # Rate configurations
    rates = {"domestic": 0.50, "commercial": 0.75, "industrial": 1.20}
    fixed_fees = {"domestic": 10.00, "commercial": 25.00, "industrial": 50.00}
    customer_fees = {"domestic": 5.00, "commercial": 12.00, "industrial": 20.00}

    c_type = customer_type.lower()
    rate = rates.get(c_type)

    if rate is None:
        return None

    ec = units * rate
    fc = fixed_fees.get(c_type)
    cc = customer_fees.get(c_type)
    ed = ec * 0.05  # Electricity Duty at 5%
    total = ec + fc + cc + ed

    return {"EC": ec, "FC": fc, "CC": cc, "ED": ed, "Total": total}

def main():
    print("--- Detailed Electricity Billing System ---")
    c_type = input("Enter Customer Type (Domestic, Commercial, Industrial): ").strip()
```

```
def main():
    print("--- Detailed Electricity Billing System ---")
    c_type = input("Enter Customer Type (Domestic, Commercial, Industrial): ").strip()
    try:
        usage = float(input("Enter units consumed: "))

        if usage < 0:
            print("Error: Units cannot be negative.")
            return

        bill = get_bill_components(usage, c_type)

        if bill:
            print(f"\n--- DETAILED BILL FOR {c_type.upper()} ---")
            print(f"Energy Charges (EC):      ${bill['EC']:>8.2f}")
            print(f"Fixed Charges (FC):       ${bill['FC']:>8.2f}")
            print(f"Customer Charges (CC):    ${bill['CC']:>8.2f}")
            print(f"Electricity Duty (ED):    ${bill['ED']:>8.2f}")
            print("-" * 35)
            print(f"TOTAL BILL AMOUNT:        ${bill['Total']:>8.2f}")
            print("-" * 35)
        else:
            print("Error: Invalid customer type.")
    except ValueError:
        print("Error: Please enter a numeric value for units.")

if __name__ == '__main__':
    main()
```

Output:

```
--- Detailed Electricity Billing System ---
Enter Customer Type (Domestic, Commercial, Industrial): Domestic
Enter units consumed: 50

--- DETAILED BILL FOR DOMESTIC ---
Energy Charges (EC):      $   25.00
Fixed Charges (FC):       $   10.00
Customer Charges (CC):    $    5.00
Electricity Duty (ED):    $    1.25
-----------------------------------
TOTAL BILL AMOUNT:        $   41.25
-----------------------------------
```

 Accuracy: The results were cross-checked with manual calculations to ensure correctness and reliability.

 Readability: The code structure was enhanced by using functions, adding comments, and presenting output in a clear, formatted manner.

 Applicability: The modular design makes the program scalable and suitable for extending into a multi-customer billing system.