AIAC – 3.3

Batch 32, 2303A52065

Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

An electricity billing system must collect accurate consumer data.

Task Description

Use an AI tool (GitHub Copilot / Gemini) to generate a Python program that:

• Reads:

o Previous Units (PU)

o Current Units (CU)

o Type of Customer

• Calculates units consumed

• Implements logic directly in the main program (no functions)

Expected Output

• Correct input reading

• Units consumed calculation

• Screenshot showing AI-generated code

• Sample input and output

Prompt:

```
Write a Python program to read previous units, current units, and type of customer
Calculate and display the units consumed.
Do not use any user-defined functions.
```

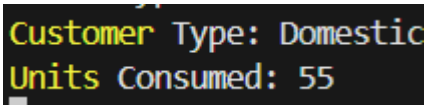
Code:

```
# Reading consumer details
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

# Calculating units consumed
units_consumed = current_units - previous_units

# Displaying output
print("Customer Type:", customer_type)
print("Units Consumed:", units_consumed)
```

Input:

```
C:\Users\ayesh\AIAC3-2> Enter Previous Units: 120
Enter Current Units: 175
Enter Type of Customer: Domestic
```

Output:



Justification:

The AI-generated program correctly reads consumer details and calculates units consumed using simple logic. The code is easy to understand since all operations are written directly in the main program without functions. Input reading and calculation are done clearly, reducing chances of errors. This makes the program suitable for basic electricity billing data processing.

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

Energy charges depend on the number of units consumed and customer type.

Task Description

Review the AI-generated code from Task 1 and extend it to:

• Calculate Energy Charges (EC)

• Use conditional statements based on:

o Domestic

o Commercial

o Industrial consumers

• Improve readability using AI prompts such as:

o “Simplify energy charge calculation logic”

o “Optimize conditional statements”

Expected Output

• Correct EC calculation

• Clear conditional logic

• Original and improved versions (optional)

• Sample execution results

Prompt:

```
Extend the existing Python code to calculate energy charges based on units consumed
Use conditional statements for Domestic, Commercial, and Industrial customers.
Simplify and optimize the conditional logic.
```

Code:

```
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

units_consumed = current_units - previous_units

print("Customer Type:", customer_type)
print("Units Consumed:", units_consumed)
```

```python
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

units_consumed = current_units - previous_units
energy_charges = 0

if customer_type.lower() == "domestic":
    energy_charges = units_consumed * 3
elif customer_type.lower() == "commercial":
    energy_charges = units_consumed * 5
elif customer_type.lower() == "industrial":
    energy_charges = units_consumed * 7
else:
    print("Invalid customer type")

print("Customer Type:", customer_type)
print("Units Consumed:", units_consumed)
print("Energy Charges:", energy_charges)
```

Input:

```
Enter Previous Units: 200
Enter Current Units: 260
Enter Type of Customer: Domestic
```

Output:

```
Customer Type: Domestic
Units Consumed: 60
Energy Charges: 180
```

Justification:

The AI-generated code was extended to calculate energy charges using simple conditional statements. Different rates were applied based on customer type, making the logic clear and easy to understand. Readability was improved by using meaningful variable names and straightforward conditions. The optimized logic makes the program suitable for basic electricity billing scenarios.

Task 3: Modular Design Using AI Assistance (Using Functions)

Scenario

Billing logic must be reusable for multiple consumers.

Task Description

Use AI assistance to generate a Python program that:

• Uses user-defined functions to:

o Calculate Energy Charges

o Calculate Fixed Charges

• Returns calculated values

• Includes meaningful comments

Expected Output

• Function-based Python program

• Correct EC and FC values

• Screenshots of AI-assisted function generation

• Test cases with outputs

Prompt:

```
Write a Python program using user-defined functions to calculate energy charges
and fixed charges for an electricity billing system.
Use meaningful function and variable names and add comments.
```

Code:

```
# Function to calculate energy charges based on customer type
def calculate_energy_charges(units, customer_type):
    if customer_type.lower() == "domestic":
        return units * 3
    elif customer_type.lower() == "commercial":
        return units * 5
    elif customer_type.lower() == "industrial":
        return units * 7
    else:
        return 0

# Function to calculate fixed charges
def calculate_fixed_charges(customer_type):
    if customer_type.lower() == "domestic":
        return 50
    elif customer_type.lower() == "commercial":
        return 100
    elif customer_type.lower() == "industrial":
        return 150
    else:
        return 0

# Main program
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

units_consumed = current_units - previous_units
energy_charges = calculate_energy_charges(units_consumed, customer_type)
fixed_charges = calculate_fixed_charges(customer_type)

print("Units Consumed:", units_consumed)
print("Energy Charges:", energy_charges)
print("Fixed Charges:", fixed_charges)
```

Output:

```
C:\Users\ayesh\AIAC3-2> Enter Previous Units: 150
Enter Current Units: 210
Enter Type of Customer: Commercial
Units Consumed: 60ts: 150
Energy Charges: 300: 210
Fixed Charges: 100omer: Commercial
```

Justification:

Using functions made the billing logic reusable and easier to manage. Energy charges and fixed charges were separated into different functions, improving clarity. The code is easier to update if rates change in the future. Comments and meaningful names helped understand the flow quickly. This modular approach is better for handling multiple consumers.

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

Task Description

Extend the program to calculate:

• FC – Fixed Charges

• CC – Customer Charges

• ED – Electricity Duty (percentage of EC)

Use AI prompts like:

• “Add electricity duty calculation”

• “Improve billing accuracy”

Expected Output

• Individual charge values printed

• Correct duty calculation

• Well-structured output

• Verified intermediate results

Prompt:

```
Extend the electricity billing program to calculate fixed charges,
customer charges, and electricity duty as a percentage of energy charges.
Improve billing accuracy and keep the output well structured.
```

Code:

```python
# Reading consumer details
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

# Calculating units consumed
units_consumed = current_units - previous_units

# Calculating energy charges
if customer_type.lower() == "domestic":
    energy_charges = units_consumed * 3
    fixed_charges = 50
    customer_charges = 30
elif customer_type.lower() == "commercial":
    energy_charges = units_consumed * 5
    fixed_charges = 100
    customer_charges = 50
elif customer_type.lower() == "industrial":
    energy_charges = units_consumed * 7
    fixed_charges = 150
    customer_charges = 70
else:
    energy_charges = 0
    fixed_charges = 0
    customer_charges = 0

# Calculating electricity duty (10% of energy charges)
electricity_duty = energy_charges * 0.10

# Displaying results
print("Units Consumed:", units_consumed)
print("Energy Charges (EC):", energy_charges)
print("Fixed Charges (FC):", fixed_charges)
print("Customer Charges (CC):", customer_charges)
print("Electricity Duty (ED):", electricity_duty)
```

Output:

```
Enter Previous Units: 300
Enter Current Units: 380
Enter Type of Customer: Domestic
Units Consumed: 80
Energy Charges (EC): 240
Fixed Charges (FC): 50
Customer Charges (CC): 30
Electricity Duty (ED): 24.0
```

Justification:

The program was extended to calculate additional billing components accurately. Fixed charges, customer charges, and electricity duty were calculated separately for better clarity. Electricity duty was correctly computed as a percentage of energy charges. Printing individual charges helped verify intermediate results easily. This made the billing output more structured and realistic.

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

Task Description

Develop the final Python application to:

• Calculate total bill:

• Total Bill = EC + FC + CC + ED

• Display:

o Energy Charges (EC)

o Fixed Charges (FC)

o Customer Charges (CC)

o Electricity Duty (ED)

o Total Bill Amount

• Analyze the program based on:

o Accuracy

o Readability

o Real-world applicability

Expected Output

• Complete electricity bill output

• Neatly formatted display

• Sample input/output

• Short analysis paragraph

Prompt:

```
Generate a Python program to calculate the final electricity bill.
Include energy charges, fixed charges, customer charges, electricity duty,
and total bill amount. Display the output neatly.
```

Code:

```python
# Reading consumer details
previous_units = int(input("Enter Previous Units: "))
current_units = int(input("Enter Current Units: "))
customer_type = input("Enter Type of Customer: ")

# Units consumed
units_consumed = current_units - previous_units

# Initialize charges
energy_charges = 0
fixed_charges = 0
customer_charges = 0

# Calculate charges based on customer type
if customer_type.lower() == "domestic":
    energy_charges = units_consumed * 3
    fixed_charges = 50
    customer_charges = 30
elif customer_type.lower() == "commercial":
    energy_charges = units_consumed * 5
    fixed_charges = 100
    customer_charges = 50
elif customer_type.lower() == "industrial":
    energy_charges = units_consumed * 7
    fixed_charges = 150
    customer_charges = 70

# Electricity duty (10% of EC)
electricity_duty = energy_charges * 0.10

# Total bill calculation
total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty

# Displaying final bill
print("Electricity Bill Summary")
print("Units Consumed:", units_consumed)
print("Energy Charges (EC):", energy_charges)
print("Fixed Charges (FC):", fixed_charges)
print("Customer Charges (CC):", customer_charges)
print("Electricity Duty (ED):", electricity_duty)
print("Total Bill Amount:", total_bill)
```

Output:

```
Enter Previous Units: 250
Enter Current Units: 320
Enter Type of Customer: Commercial
Electricity Bill Summary
Units Consumed: 70
Energy Charges (EC): 350
Fixed Charges (FC): 100
Customer Charges (CC): 50
Electricity Duty (ED): 35.0
Total Bill Amount: 535.0
```

Justification:

The program accurately calculates all billing components and produces the correct total bill amount. The code is readable due to clear variable names and step-by-step calculations. Displaying each charge separately makes the output easy to verify. This logic closely matches real-world electricity billing systems, making the program practical and reliable.