# AI ASSISTED CODING

Prompt Engineering-Improving Prompts

**H.no:2403a510G7** **Name:N.Prudhvi**

**Batch-06**

**TASK-1:**

PROMPT:

"Write a Python function to check if a number is prime."

CODE:

```python
def is_prime(n: int) -> bool:
    """Check if a number is prime."""
    if n <= 1:
        return False
    if n <= 3:
        return True
    if n % 2 == 0 or n % 3 == 0:
        return False

    # check divisibility up to sqrt(n)
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return False
        i += 6
    return True
```

ONE-SHOT:

```python
def check_prime(n: int) -> str:
    """Check if a number is prime and return 'Prime'
    if n <= 1:
        return "Not Prime"
    if n <= 3:
        return "Prime"
    if n % 2 == 0 or n % 3 == 0:
        return "Not Prime"
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return "Not Prime"
        i += 6
    return "Prime"

# Example usage:
print("Input: 5 → Output:", check_prime(5))   # Prime
```

OUTPUT:

```
Input: 5 → Output: Prime
```

FEW SHOT:

```python
def check_prime(n: int) -> str:
    """Check if a number is prime and return 'Prime' or '
    if n <= 1:
        return "Not Prime"
    if n <= 3:
        return "Prime"
    if n % 2 == 0 or n % 3 == 0:
        return "Not Prime"
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return "Not Prime"
        i += 6
    return "Prime"

# Example usage:
print("Input: 7 → Output:", check_prime(7))   # Prime
print("Input: 10 → Output:", check_prime(10)) # Not Prime
print("Input: 2 → Output:", check_prime(2)) # Prime
```

OUTPUT:

```
Input: 7 → Output: Prime
Input: 10 → Output: Not Prime
Input: 2 → Output: Prime
```

CONTEXT MANAGED:

```python
def check_prime(n):
    if n < 0:
        return "Not Prime"
    if n in (0, 1):
        return "Neither Prime Nor Composite"
    if n <= 3:
        return "Prime"
    if n % 2 == 0 or n % 3 == 0:
        return "Not Prime"
    i = 5
    while i * i <= n:
        if n % i == 0 or n % (i + 2) == 0:
            return "Not Prime"
        i += 6
    return "Prime"


# Example usage with output
number = int(input("Enter a number: "))
print(check_prime(number))
```

**TASK-2**:

PROMPT:

Develop the logic to calculate data charges (DC), value-added charges (VC), and applicable taxes based on the user inputs and a realistic charge structure.

**Reasoning**: Define the charge structure and create a function to calculate data charges, value-added charges, and tax based on the provided inputs and charge structure.

CODE:

```
# 1. Define a realistic charge structure
CHARGE_STRUCTURE = {
    "data_rates_per_gb": {
        "Prepaid": 10.0,  # Cost per GB for Prepaid
        "Postpaid": 8.0   # Cost per GB for Postpaid
    },
    "additional_service_charges": {
        "International Calls": 15.0,
        "Premium SMS": 5.0,
        "Music Streaming": 12.0
    },
    "tax_rate": 0.15 # 15% tax rate
}

# 2. Create a function to calculate charges
def calculate_bill(data_consumed, plan_type, additional_services):
    """
    Calculates mobile data charges, value-added charges, and tax.

    Args:
        data_consumed: The amount of data consumed in GB (float).
        plan_type: The plan type ('Prepaid' or 'Postpaid') (string).
        additional_services: A list of additional services used (list of strings).

    Returns:
        A dictionary containing data_charge, value_added_charge, tax_amount, and subtotal.
    """
    # 3. Calculate data charge
    if plan_type in CHARGE_STRUCTURE["data_rates_per_gb"]:
        data_charge = data_consumed * CHARGE_STRUCTURE["data_rates_per_gb"][plan_type]
    else:
        return {"error": "Invalid plan type"}
```

```
    # 4. Calculate value-added charges
    value_added_charge = 0
    for service in additional_services:
        if service in CHARGE_STRUCTURE["additional_service_charges"]:
            value_added_charge += CHARGE_STRUCTURE["additional_service_charges"][service]
        else:
            # Optionally handle unknown services, e.g., log a warning or add a small default charge
            pass # For simplicity, we'll ignore unknown services here

    # Calculate subtotal before tax
    subtotal = data_charge + value_added_charge

    # 5. Calculate tax amount
    tax_amount = subtotal * CHARGE_STRUCTURE["tax_rate"]

    # 6. Return the calculated charges
    return {
        "data_charge": data_charge,
        "value_added_charge": value_added_charge,
        "subtotal": subtotal,
        "tax_amount": tax_amount,
        "total_bill": subtotal + tax_amount
    }

# Example usage (for testing)
example_bill = calculate_bill(10.5, "Postpaid", ["International Calls", "Music Streaming"])
print(example_bill)

example_bill_prepaid = calculate_bill(5, "Prepaid", ["Premium SMS"])
print(example_bill_prepaid)

example_bill_invalid_plan = calculate_bill(10, "Unknown", [])
print(example_bill_invalid_plan)
```

```
def display_bill(bill_details, data_consumed, plan_type, additional_services):
    """
    Displays an itemized mobile data bill.

    Args:
        bill_details: A dictionary containing calculated bill components
                      (data_charge, value_added_charge, subtotal, tax_amount, total_bill).
        data_consumed: The amount of data consumed in GB (float).
        plan_type: The plan type ('Prepaid' or 'Postpaid') (string).
        additional_services: A list of additional services used (list of strings).
    """
    print("----------------------------------------")
    print("          MOBILE DATA BILL")
    print("----------------------------------------")
    print(f"Plan Type: {plan_type}")
    print(f"Data Usage: {data_consumed:.2f} GB")
    print(f"Data Charge: ${bill_details['data_charge']:.2f}")
    print("\nAdditional Services:")
    if additional_services:
        for service in additional_services:
            # Assuming we can access the individual service charge from the global structure
            # In a real app, this might be passed in bill_details or looked up again
            service_charge = CHARGE_STRUCTURE["additional_service_charges"].get(service, 0.0)
            print(f"- {service}: ${service_charge:.2f}")
    else:
        print("  None")

    print("----------------------------------------")
    print(f"Subtotal: ${bill_details['subtotal']:.2f}")
    print(f"Tax Amount: ${bill_details['tax_amount']:.2f}")
    print("----------------------------------------")
    print(f"Total Bill: ${bill_details['total_bill']:.2f}")
    print("----------------------------------------")
```

```
    print("----------------------------------------")
    print(f"Subtotal: ${bill_details['subtotal']:.2f}")
    print(f"Tax Amount: ${bill_details['tax_amount']:.2f}")
    print("----------------------------------------")
    print(f"Total Bill: ${bill_details['total_bill']:.2f}")
    print("----------------------------------------")

# Example usage (assuming calculate_bill is defined and CHARGE_STRUCTURE is available)
example_bill_details = calculate_bill(10.5, "Postpaid", ["International Calls", "Music Streaming"])
display_bill(example_bill_details, 10.5, "Postpaid", ["International Calls", "Music Streaming"])

example_bill_details_prepaid = calculate_bill(5, "Prepaid", ["Premium SMS"])
display_bill(example_bill_details_prepaid, 5, "Prepaid", ["Premium SMS"])

example_bill_details_no_services = calculate_bill(2.3, "Prepaid", [])
display_bill(example_bill_details_no_services, 2.3, "Prepaid", [])
```

**TASK-3:**

CODE:

```
# 1. Create a Python dictionary named LPG_PRICE_LIST
LPG_PRICE_LIST = {
    "Domestic 14.2 kg": 905.00,
    "Domestic 5 kg": 335.50,
    "Commercial 19 kg": 1886.50,
    "Commercial 47.5 kg": 4712.00
}

# 2. Define variables for minimum and maximum delivery charges
MIN_DELIVERY_CHARGE = 10
MAX_DELIVERY_CHARGE = 50

# 3. Print the definitions to verify
print("LPG Price List:")
print(LPG_PRICE_LIST)
print(f"\nMinimum Delivery Charge: ₹{MIN_DELIVERY_CHARGE}")
print(f"Maximum Delivery Charge: ₹{MAX_DELIVERY_CHARGE}")
```

```
LPG Price List:
{'Domestic 14.2 kg': 905.0, 'Domestic 5 kg': 335.5, 'Commercial 19 kg': 1886.5, 'Commercial 47.5 kg': 4712.0}

Minimum Delivery Charge: ₹10
Maximum Delivery Charge: ₹50
```

```python
def calculate_lpg_bill(cylinder_type, num_cylinders, subsidy_amount):
    """
    Calculates the total LPG bill based on cylinder type, number of cylinders,
    delivery charges, and applicable subsidy.

    Args:
        cylinder_type: The type of LPG cylinder (string).
        num_cylinders: The number of cylinders booked (integer).
        subsidy_amount: The subsidy amount applicable for domestic cylinders (float

    Returns:
        A dictionary containing the calculated bill details, or an error message.
    """
    # 2. Retrieve the price per cylinder
    price_per_cylinder = LPG_PRICE_LIST.get(cylinder_type)

    if price_per_cylinder is None:
        return {"error": f"Invalid cylinder type: {cylinder_type}"}

    # 3. Calculate the subtotal
    subtotal = price_per_cylinder * num_cylinders

    # 4. Generate a random delivery charge
    delivery_charge = random.randint(MIN_DELIVERY_CHARGE, MAX_DELIVERY_CHARGE)

    # 5. Calculate the total cost before subsidy
    total_cost_before_subsidy = subtotal + delivery_charge

    # 6. Apply subsidy if applicable
    subsidy_applied = 0
    if cylinder_type in ["Domestic 14.2 kg", "Domestic 5 kg"]:
        subsidy_applied = subsidy_amount
        total_cost_after_subsidy = total_cost_before_subsidy - subsidy_applied
        # Ensure the total cost does not go below zero
        total_bill = max(0, total_cost_after_subsidy)
    else:
        total_bill = total_cost_before_subsidy

    # 7. Return the calculated values
    return {
        "cylinder_type": cylinder_type,
        "num_cylinders": num_cylinders,
        "price_per_cylinder": price_per_cylinder,
        "subtotal": subtotal,
        "delivery_charge": delivery_charge,
        "subsidy_amount_provided": subsidy_amount,
        "subsidy_applied": subsidy_applied,
        "total_bill": total_bill
    }

# Example usage (for testing)
# print(calculate_lpg_bill("Domestic 14.2 kg", 2, 200.0))
# print(calculate_lpg_bill("Commercial 19 kg", 1, 0.0))
# print(calculate_lpg_bill("Domestic 5 kg", 3, 50.0))
# print(calculate_lpg_bill("Invalid Type", 1, 100.0))
```

```
 ---------------------------------
           LPG Bill Details
 ---------------------------------
 Cylinder Type: Commercial 19 kg
 Number of Cylinders: 1
 Price per Cylinder: ₹1886.50
 ---------------------------------
 Subtotal: ₹1886.50
 Delivery Charge: ₹27.00
 Subsidy Applied: ₹0.00
 ---------------------------------
 Total Bill Amount: ₹1913.50
 ---------------------------------
```

```python
def lpg_bill_calculator():
    print("------ LPG Bill Calculator ------")

    # Cylinder price list
    prices = {
        "Domestic 14.2 kg": 905.00,
        "Domestic 5 kg": 335.50,
        "Commercial 19 kg": 1886.50,
        "Commercial 47.5 kg": 4712.00
    }
    # Step 1: Take user input for cylinder type
    print("\nSelect Cylinder Type:")
    for i, cylinder in enumerate(prices.keys(), start=1):
        print(f"{i}. {cylinder}")

    choice = int(input("Enter your choice (1-4): "))
    cylinder_type = list(prices.keys())[choice - 1]
    price_per_cylinder = prices[cylinder_type]
```

```python
# Step 2: Number of cylinders
num_cylinders = int(input("\nEnter number of cylinders booked: "))

# Step 3: Subsidy (only for Domestic)
subsidy = 0.0
if "Domestic" in cylinder_type:
    subsidy = float(input("Enter subsidy amount per cylinder: "))

# Step 4: Delivery Charges
delivery_charges = float(input("Enter delivery charges (₹10 to ₹50): "))

# Step 5: Billing formula
base_amount = price_per_cylinder * num_cylinders
total_subsidy = subsidy * num_cylinders if "Domestic" in cylinder_type else
bill_amount = base_amount - total_subsidy + delivery_charges
```

```python
    # Step 6: Display itemized bill
    print("\n------ ITEMIZED LPG BILL ------")
    print(f"Cylinder Type       : {cylinder_type}")
    print(f"Number of Cylinders : {num_cylinders}")
    print(f"Price per Cylinder  : ₹{price_per_cylinder}")
    print(f"Base Amount         : ₹{base_amount:.2f}")
    print(f"Subsidy             : -₹{total_subsidy:.2f}")
    print(f"Delivery Charges    : ₹{delivery_charges:.2f}")
    print(f"-----------------------------")
    print(f"Total Bill Amount   : ₹{bill_amount:.2f}")

lpg_bill_calculator()
```

OUTPUT:

```
Select Cylinder Type:
1. Domestic 14.2 kg
2. Domestic 5 kg
3. Commercial 19 kg
4. Commercial 47.5 kg
Enter your choice (1-4): 3

Enter number of cylinders booked: 2
Enter delivery charges (₹10 to ₹50): 20
```

```
------ ITEMIZED LPG BILL ------
Cylinder Type       : Commercial 19 kg
Number of Cylinders : 2
Price per Cylinder  : ₹1886.5
Base Amount         : ₹3773.00
Subsidy             : -₹0.00
Delivery Charges    : ₹20.00
-------------------------------
Total Bill Amount   : ₹3793.00
```