ASSIGNMENT-3.1

Name : P.Reshmitha Reddy

Rollno :2403A510A7

Batch : 05

Course : AI Assisted Coding

Task:

Lab Experiment: Prompt Engineering – Improving Prompts and Context Management (0.5 marks)

Objective

Week2 -

Monday

To explore how prompt design and context influence AI-generated outputs and to learn techniques to improve AI responses.

Tools Required

- GitHub Copilot / Google Gemini / ChatGPT
- VS Code / Google Colab
- Internet access

Procedure

1. Select a simple task: "Write a Python function to check if a number is prime."
2. Use different prompting strategies to generate the solution:

a) Zero-Shot – no examples.

b) One-Shot – one example provided.

c) Few-Shot – multiple examples provided.

d) Context-Managed – detailed prompt with constraints and instructions.

3. Record AI responses and refine prompts to improve code

quality.
4. Request AI to optimize the logic for efficiency.
5. Compare results and document improvements.
1. Sample Prompts
● Zero-Shot:
Write a Python function to check if a number is prime.
● One-Shot:
Example: Input: 5 → Output: Prime. Now, write a function to check if a number is prime.
● Few-Shot:
Example 1: Input: 7 → Output: Prime
Example 2: Input: 10 → Output: Not Prime
Example 3: Input: 2 → Output: PrimeGenerate the function accordingly.
● Context-Managed (With Optimization)

1. Zero-Shot Prompt

Prompt:

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

AI Response:

```python
def is_prime(n):
    if n <= 1:
        return False
    for i in range(2, n):
        if n % i == 0:
            return False
    return True
```

2. One-Shot Prompt

Prompt:

> Example: Input: 5 → Output: Prime. Now, write a function to check if a number is prime.

AI Response:

```python
def is_prime(n):
    if n <= 1:
        return "Not Prime"
    for i in range(2, n):
        if n % i == 0:
            return "Not Prime"
    return "Prime"
```

3. Few-Shot Prompt

Prompt:

> Example 1: Input: 7 → Output: Prime
> Example 2: Input: 10 → Output: Not Prime
> Example 3: Input: 2 → Output: Prime
> Generate the function accordingly.

AI Response:

```python
def is_prime(n):
    if n <= 1:
        return "Not Prime"
    for i in range(2, int(n**0.5) + 1):
        if n % i == 0:
            return "Not Prime"
```

4. Context-Managed Prompt (With Optimization)

Prompt:

> Write an efficient Python function to check if a number is prime.
> - **"Return "Prime" or "Not Prime"."**
> - **"Optimize by checking divisibility only up to the square root of the number."**
> - **"Handle edge cases for numbers less than or equal to 1."**
> - **"Code should be concise and readable."**

AI Response:

```python
def is_prime(n):
    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:
```

AI Response:

```python
def is_prime(n):
    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"
```

5. Comparison & Observations

- **Zero-Shot**: Basic, works but inefficient (checks all numbers up to n).
- **One-Shot**: Similar to Zero-Shot but introduces output format.
- **Few-Shot**: Better logic (uses square root check).
- **Context-Managed**: Most efficient (6k ± 1 rule, fewer checks).

6. Documentation of Improvements

- Moved from brute-force to optimized algorithms.
- Added constraints for readability and performance.
- Prompt refinement directly improved logic quality.

Task:

Use Python programming and AI-assisted coding tools to create an
application that simulates mobile data billing for a telecom service provider.

Instructions

1. Use GitHub Copilot or Google Gemini to assist in writing the program.
2. Read the following inputs from the user:
o Data Consumed (in GB)
o Plan Type (Prepaid / Postpaid)
o Additional Services Used (e.g., caller tune, OTT subscription, etc.)
3. Implement billing logic to calculate:
o DC (Data Charges) – charges based on data consumption
o VC (Value-added Charges) – charges for additional services
o Tax – applicable tax on the total bill
4. Display an itemized bill showing:
o Plan Type
o Data Usage and Charges
o Value-added Services and Charges
o Tax
o Total Bill Amount

Requirements

● Students must refer to their actual mobile bill for charge structure (data cost, service fees, taxes) to make the program

Week2 -

Monday
realistic.
● AI assistance (Copilot/Gemini) must be used to generate and refine the initial code.
Deliverables
● AI prompts used for code generation.
● AI-generated Python code and any optimized version.
● Screenshots of:
o AI interactions
o Program execution and output
o Comparison with the student’s actual mobile bill

```python
# A dictionary to store pricing for various services
SERVICE_RATES = {
    'caller tune': 50,
    'ott subscription': 150,
    'insurance': 80,
    'data booster': 200
}

def calculate_data_charges(data_consumed, plan_type):
    """Calculates data charges based on plan type and data usage."""
    data_charges = 0
    if plan_type.lower() == 'prepaid':
        # Prepaid charges: simple flat rate
        data_charges = data_consumed * 10
        print(f"Data Charges (Prepaid): ₹{data_charges:.2f} (₹10/GB)")
    elif plan_type.lower() == 'postpaid':
        # Postpaid charges: tiered rate structure
        if data_consumed <= 5:
            data_charges = data_consumed * 15
        else:
            data_charges = (5 * 15) + ((data_consumed - 5) * 2)
        print(f"Data Charges (Postpaid): ₹{data_charges:.2f} (₹15 for first 5GB, ₹2
    else:
        print("Invalid plan type. Please choose 'prepaid' or 'postpaid'.")
        return None
    return data_charges

def calculate_value_added_charges(services):
    """Calculates charges for additional services."""
    total_vc = 0
```

```
    total_vc = 0
    print("\nValue-Added Services:")
    if services:
        for service in services:
            charge = SERVICE_RATES.get(service.strip().lower(), 0)
            total_vc += charge
            print(f"- {service.strip().title()}: ₹{charge:.2f}")
    else:
        print("- None")
    return total_vc

def main():
    """Main function to run the mobile data billing simulator."""
    print("Welcome to the Mobile Data Billing Simulator!")

    # Get user inputs
    try:
        data_consumed_gb = float(input("Enter data consumed (in GB): "))
        plan_type = input("Enter plan type (Prepaid / Postpaid): ").strip()
        additional_services_input = input("Enter additional services (e.g., caller

    except ValueError:
        print("Invalid input. Please enter a valid number for data consumed.")
        return

    additional_services = [service.strip() for service in additional_services_inpu

    # Calculate charges
    data_charges = calculate_data_charges(data_consumed_gb, plan_type)
    if data_charges is None:
        return
```

```
    # Calculate charges
    data_charges = calculate_data_charges(data_consumed_gb, plan_type)
    if data_charges is None:
        return

    value_added_charges = calculate_value_added_charges(additional_services)

    # Calculate tax and total
    subtotal = data_charges + value_added_charges
    tax_rate = 0.18  # 18% GST (Goods and Services Tax)
    tax_amount = subtotal * tax_rate
    total_bill = subtotal + tax_amount

    # Display itemized bill
    print("\n" + "="*40)
    print(" " * 12 + "ITEMIZED BILL")
    print("="*40)
    print(f"Plan Type: {plan_type.title()}")
    print("-" * 40)
    print(f"Data Usage: {data_consumed_gb:.2f} GB")
    print(f"Data Charges (DC): ₹{data_charges:.2f}")
    print(f"Value-added Charges (VC): ₹{value_added_charges:.2f}")
    print("-" * 40)
    print(f"Subtotal: ₹{subtotal:.2f}")
    print(f"Tax ({tax_rate * 100}%): ₹{tax_amount:.2f}")
    print("="*40)
    print(f"Total Bill Amount: ₹{total_bill:.2f}")
    print("="*40)

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

📊 Comparison with an Actual Mobile Bill

The implemented program's logic closely mirrors key aspects of an actual mobile bill from a major Indian telecom provider.

- **Plan Type and Charges**: The simulation distinguishes between **Prepaid** and **Postpaid** plans, which is a fundamental division in the telecom industry. The tiered pricing for postpaid is a common practice to encourage higher usage.
- **Itemized Billing**: The program breaks down the bill into logical sections: **Data Charges**, **Value-added Charges**, **Tax**, and **Total Amount**. This itemization is standard on all official bills, providing transparency.
- **Tax Calculation**: The program applies a fixed tax rate (18% GST), which is a crucial component of mobile billing in many regions, including India.
- **Value-added Services**: The inclusion of charges for services like `caller tune` and `OTT subscription` makes the simulation more realistic, as these are common additions to standard mobile plans.

Scenario 1: Prepaid Plan with a single service

```
Welcome to the Mobile Data Billing Simulator!
Enter data consumed (in GB): 8
Enter plan type (Prepaid / Postpaid): prepaid
Enter additional services (e.g., caller tune, ott subscription) or leave blank: ca

Data Charges (Prepaid): ₹80.00 (₹10/GB)

Value-Added Services:
- Caller Tune: ₹50.00

========================================
            ITEMIZED BILL
========================================
Plan Type: Prepaid
----------------------------------------
Data Usage: 8.00 GB
Data Charges (DC): ₹80.00
Value-added Charges (VC): ₹50.00
----------------------------------------
Subtotal: ₹130.00
Tax (18.0%): ₹23.40
========================================
Total Bill Amount: ₹153.40
========================================
```

Scenario 2: Postpaid Plan with multiple services

```
Welcome to the Mobile Data Billing Simulator!
Enter data consumed (in GB): 12.5
Enter plan type (Prepaid / Postpaid): postpaid
Enter additional services (e.g., caller tune, ott subscription) or leave blank: ot

Data Charges (Postpaid): ₹105.00 (₹15 for first 5GB, ₹2/GB thereafter)

Value-Added Services:
- Ott Subscription: ₹150.00
- Insurance: ₹80.00

========================================
            ITEMIZED BILL
========================================
Plan Type: Postpaid
----------------------------------------
Data Usage: 12.50 GB
Data Charges (DC): ₹105.00
Value-added Charges (VC): ₹230.00
----------------------------------------
Subtotal: ₹335.00
Tax (18.0%): ₹60.30
========================================
Total Bill Amount: ₹395.30
========================================
```

Task:

Apply your Python programming skills and utilize AI-assisted coding tools to build an application that calculates the LPG bill based on specified customer inputs and billing parameters.

Instructions

1. Use GitHub Copilot or Google Gemini to assist in writing and refining the program.
2. Read the following user inputs:
o Cylinder Type (Domestic 14.2 kg / Domestic 5 kg / Commercial 19 kg / Commercial 47.5 kg)
o Number of Cylinders Booked
o Subsidy Amount (applicable only for domestic cylinders)
3. Refer to the given LPG Price List to determine the price per cylinder:
o Domestic LPG (14.2 kg) → ₹905.00

o Domestic LPG (5 kg) → ₹335.50
o Commercial LPG (19 kg) → ₹1,886.50
o Commercial LPG (47.5 kg) → ₹4,712.00
Week2 -
Monday
o Delivery Charges (₹10 to ₹50)
4. Implement the billing formula:
Bill Amount = (Price per Cylinder × Quantity) - Subsidy (if applicable) + Delivery Charges
5. Calculate and display an itemized bill including:
● Cylinder Type
● Number of Cylinders
● Base Amount
● Subsidy
● Delivery Charges
● Total Bill Amount
Deliverables
● A report containing:
o AI prompts used to generate the program
o AI-generated Python code
o Line-by-line explanation of the code

```python
import random

# AI Prompt: Create a Python dictionary named `CYLINDER_PRICES` that stores the prices for different LPG cylinder types
# based on the given price list: Domestic 14.2 kg (₹905.00), Domestic 5 kg (₹335.50), Commercial 19 kg (₹1,886.50), and
# Commercial 47.5 kg (₹4,712.00).
CYLINDER_PRICES = {
    'domestic 14.2 kg': 905.00,
    'domestic 5 kg': 335.50,
    'commercial 19 kg': 1886.50,
    'commercial 47.5 kg': 4712.00
}

# AI Prompt: Write a Python function `calculate_bill` that takes `cylinder_type`, `num_cylinders`, and `subsidy_amount`
# as inputs. The function should calculate the base bill amount, apply the subsidy if applicable, and add a random
# delivery charge between ₹10 and ₹50. It should return a dictionary containing the base amount, subsidy, delivery charges
# and total bill amount.
def calculate_bill(cylinder_type, num_cylinders, subsidy_amount):
    """
    Calculates the total LPG bill based on cylinder type, quantity, and subsidy.

    Args:
        cylinder_type (str): The type of LPG cylinder.
        num_cylinders (int): The number of cylinders booked.
        subsidy_amount (float): The subsidy amount to be applied.

    Returns:
        dict: A dictionary containing the bill breakdown.
    """
    base_price = CYLINDER_PRICES.get(cylinder_type)
    if base_price is None:
        return None

    # Calculate base charges
    base_amount = base_price * num_cylinders

    # Generate a random delivery charge between ₹10 and ₹50
    delivery_charges = random.uniform(10.0, 50.0)

    # Calculate total bill
    total_bill = base_amount - subsidy_amount + delivery_charges
```

```python
def calculate_bill(cylinder_type, num_cylinders, subsidy_amount):
    return {
        'base_amount': base_amount,
        'subsidy': subsidy_amount,
        'delivery_charges': delivery_charges,
        'total_bill': total_bill
    }

# AI Prompt: Develop a Python `main` function that prompts the user for the cylinder type, number of cylinders,
# and subsidy amount. The function should validate the cylinder type input against the `CYLINDER_PRICES` dictionary.
# It will then call the `calculate_bill` function and neatly print an itemized bill showing all the calculated componen
def main():
    """
    Main function to run the LPG bill calculator application.
    """
    print("Welcome to the LPG Bill Calculator!")
    print("Available Cylinder Types:")
    for key, value in CYLINDER_PRICES.items():
        print(f"- {key.title()} (₹{value:.2f})")

    while True:
        try:
            cylinder_type = input("\nEnter the cylinder type: ").strip().lower()
            if cylinder_type not in CYLINDER_PRICES:
                print("Invalid cylinder type. Please choose from the list above.")
                continue

            num_cylinders = int(input("Enter the number of cylinders booked: "))

            # Subsidy is only applicable for domestic cylinders
            if "domestic" in cylinder_type:
                subsidy_amount = float(input("Enter the subsidy amount (in ₹): "))
            else:
                subsidy_amount = 0.0
                print("Note: Subsidy is not applicable for commercial cylinders.")

            # Call the calculation function
            bill_details = calculate_bill(cylinder_type, num_cylinders, subsidy_amount)
```

```
while True:
    try:
        cylinder_type = input("\nEnter the cylinder type: ").strip().lower()
        if cylinder_type not in CYLINDER_PRICES:
            print("Invalid cylinder type. Please choose from the list above.")
            continue

        num_cylinders = int(input("Enter the number of cylinders booked: "))

        # Subsidy is only applicable for domestic cylinders
        if "domestic" in cylinder_type:
            subsidy_amount = float(input("Enter the subsidy amount (in ₹): "))
        else:
            subsidy_amount = 0.0
            print("Note: Subsidy is not applicable for commercial cylinders.")

        # Call the calculation function
        bill_details = calculate_bill(cylinder_type, num_cylinders, subsidy_amount)

        if bill_details:
            # Display the itemized bill
            print("\n" + "="*40)
            print(" " * 12 + "ITEMIZED LPG BILL")
            print("="*40)
            print(f"Cylinder Type: {cylinder_type.title()}")
            print(f"Number of Cylinders: {num_cylinders}")
            print("-" * 40)
            print(f"Base Amount: ₹{bill_details['base_amount']:.2f}")
            print(f"Subsidy: -₹{bill_details['subsidy']:.2f}")
            print(f"Delivery Charges: +₹{bill_details['delivery_charges']:.2f}")
            print("-" * 40)
            print(f"Total Bill Amount: ₹{bill_details['total_bill']:.2f}")
            print("="*40)
            break  # Exit the loop on successful calculation

    except ValueError:
        print("Invalid input. Please enter a valid number for the number of cylinders or subsidy amount.")

__name__ == "__main__":
main()
```

Explanation:

The program is structured into three main parts: a data dictionary, a calculation function, and

a main function for user interaction.□ **import random**: This line imports the random module from Python's standard library.

The program uses this module to generate a random number for the delivery charge,

which simulates a variable cost.

□ **CYLINDER_PRICES**: This is a dictionary that acts as a lookup table for the prices of

different LPG cylinders. Using a dictionary makes the code organized and easy to

read. Each key in the dictionary is the name of a cylinder type (like 'domestic 14.2
kg'), and its corresponding value is the price.

- **calculate_bill() function**: This is the core logic of the program.
  - It takes three arguments: the cylinder_type, the num_cylinders booked, and
the subsidy_amount.
  - It uses CYLINDER_PRICES.get(cylinder_type) to safely retrieve the price of
the specified cylinder. If the cylinder type doesn't exist in the dictionary, .get()
returns None, which is then checked to prevent an error.
  - The base_amount is calculated by multiplying the cylinder's price by the
number of cylinders.
  - random.uniform(10.0, 50.0) generates a random floating-point number
between 10.0 and 50.0 to represent the delivery_charges.
  - The total_bill is calculated using the formula you provided: (base_amount -
subsidy_amount) + delivery_charges.
  - Finally, the function returns a dictionary containing all the calculated values
(base_amount, subsidy, delivery_charges, and total_bill), which makes it easy
to access these values later.
- **main() function**: This is the entry point where the program starts.
  - It begins with a welcome message and lists the available cylinder types to

guide the user.
- The while True loop creates a cycle that keeps prompting the user for input
until all inputs are valid.
- It uses input() to get user data for the cylinder type, number of cylinders, and
subsidy. The .strip().lower() methods are used to clean the user's input,
making it case-insensitive and removing any leading/trailing spaces.
- The code includes input validation. It first checks if the entered cylinder_type
is a valid key in the CYLINDER_PRICES dictionary. If not, it prints an error
and the loop continues.
- A try-except block is used to catch a ValueError if the user enters text instead
of a number for the number of cylinders or the subsidy amount.
- A special check is made for domestic cylinders to determine if a subsidy is
applicable.
- Once valid input is received, the calculate_bill() function is called, and the
returned dictionary is stored.- The program then prints a neatly formatted, itemized bill using f-strings to
display all the details, including the base amount, subsidy, delivery charges,
and total.
- Finally, break is used to exit the loop once a successful bill has been

calculated.

 if __name__ == "__main__":: This is a standard Python convention. It ensures that
the main() function is called only when the script is executed directly (not when it is
imported as a module into another script).