ASSIGNMENT 3.3

NAME: METTU SEEVLA VARSHITH REDDY

H.NO: 2403A51289

SUBJECT: AI ASSISTANT CODING

BATCH: 01

TASK DESCRIPTION 1:

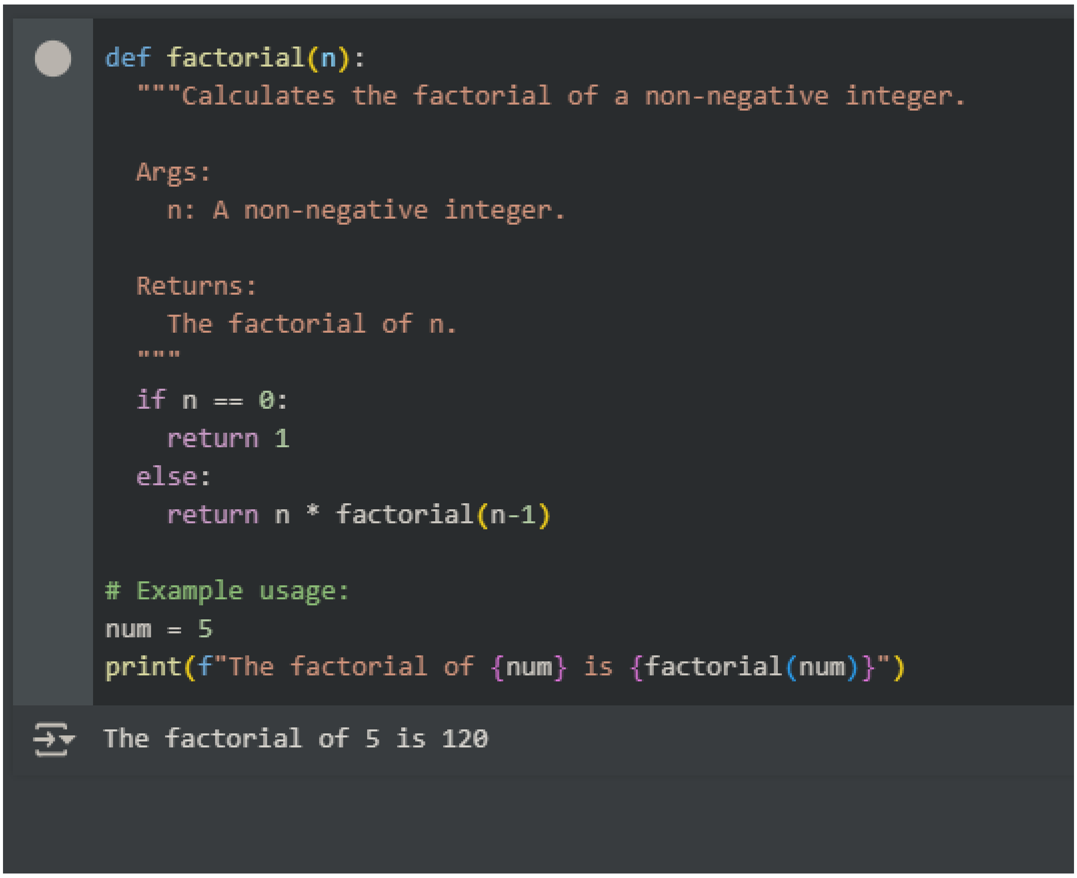
QUESTION:

Try 3 different prompts to generate a factorial function

PROMPT1:

generate a code in python factorial function

CODE WITH OUTPUT:



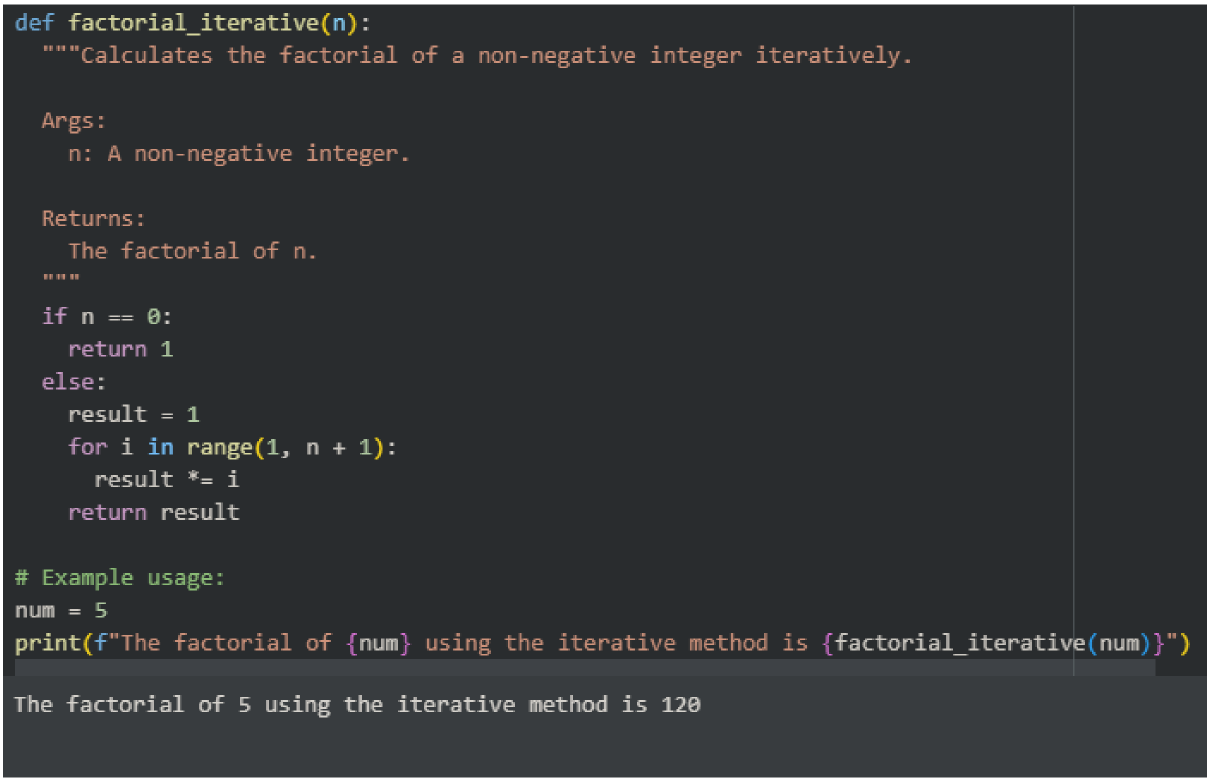
```python
def factorial(n):
  """Calculates the factorial of a non-negative integer.

  Args:
    n: A non-negative integer.

  Returns:
    The factorial of n.
  """
  if n == 0:
    return 1
  else:
    return n * factorial(n-1)

# Example usage:
num = 5
print(f"The factorial of {num} is {factorial(num)}")
```

```
The factorial of 5 is 120
```

PROMPT2:

Write an efficient code for factorial function

CODE WITH OUTPUT:



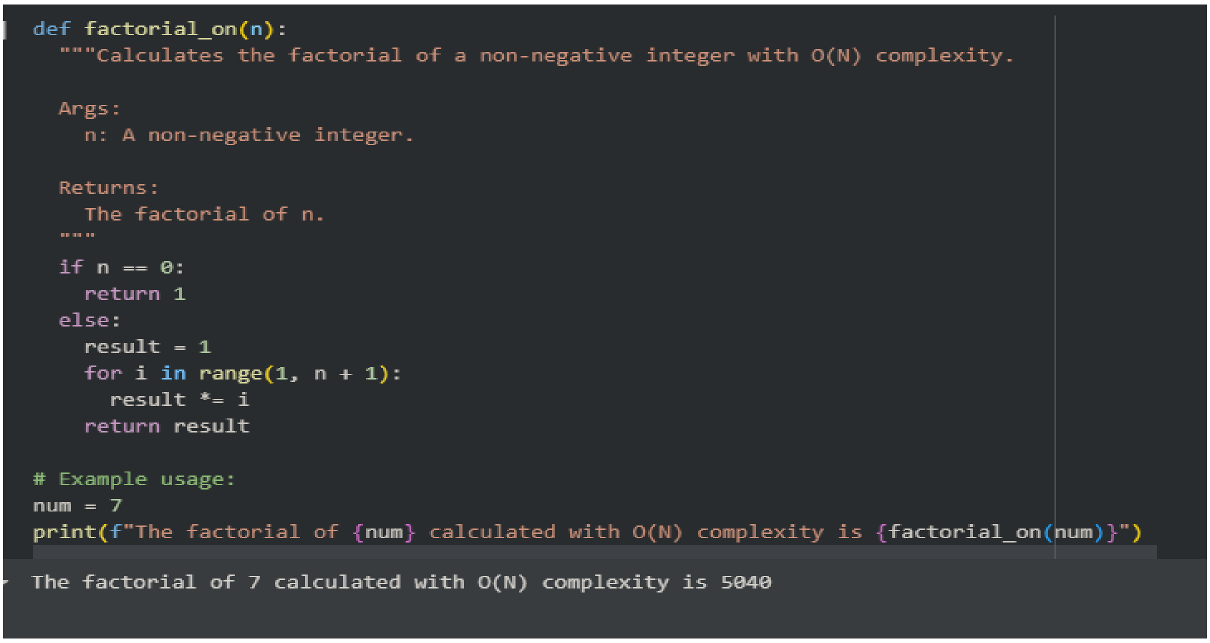
```python
def factorial_iterative(n):
  """Calculates the factorial of a non-negative integer iteratively.

  Args:
    n: A non-negative integer.

  Returns:
    The factorial of n.
  """
  if n == 0:
    return 1
  else:
    result = 1
    for i in range(1, n + 1):
      result *= i
    return result

# Example usage:
num = 5
print(f"The factorial of {num} using the iterative method is {factorial_iterative(num)}")
```

```
The factorial of 5 using the iterative method is 120
```

PROMPT3:

write a code on factorial in different approach with O(N)

CODE WITH OUTPUT:



```python
def factorial_on(n):
  """Calculates the factorial of a non-negative integer with O(N) complexity.

  Args:
    n: A non-negative integer.

  Returns:
    The factorial of n.
  """
  if n == 0:
    return 1
  else:
    result = 1
    for i in range(1, n + 1):
      result *= i
    return result

# Example usage:
num = 7
print(f"The factorial of {num} calculated with O(N) complexity is {factorial_on(num)}")
```

```
The factorial of 7 calculated with O(N) complexity is 5040
```

EXPECTED OUTPUT:

Comparison of AI-generated code styles

OBSERVATION:

All three valid implementations of the factorial function produced the correct results for the examples we tested. However, the iterative approach is generally preferred in practice for its efficiency and avoidance of potential stack overflow issues with large inputs. The iterative versions are essentially the same in terms of their core logic and O(N) time complexity.

TASK DESCRIPTION 2:

QUESTION:

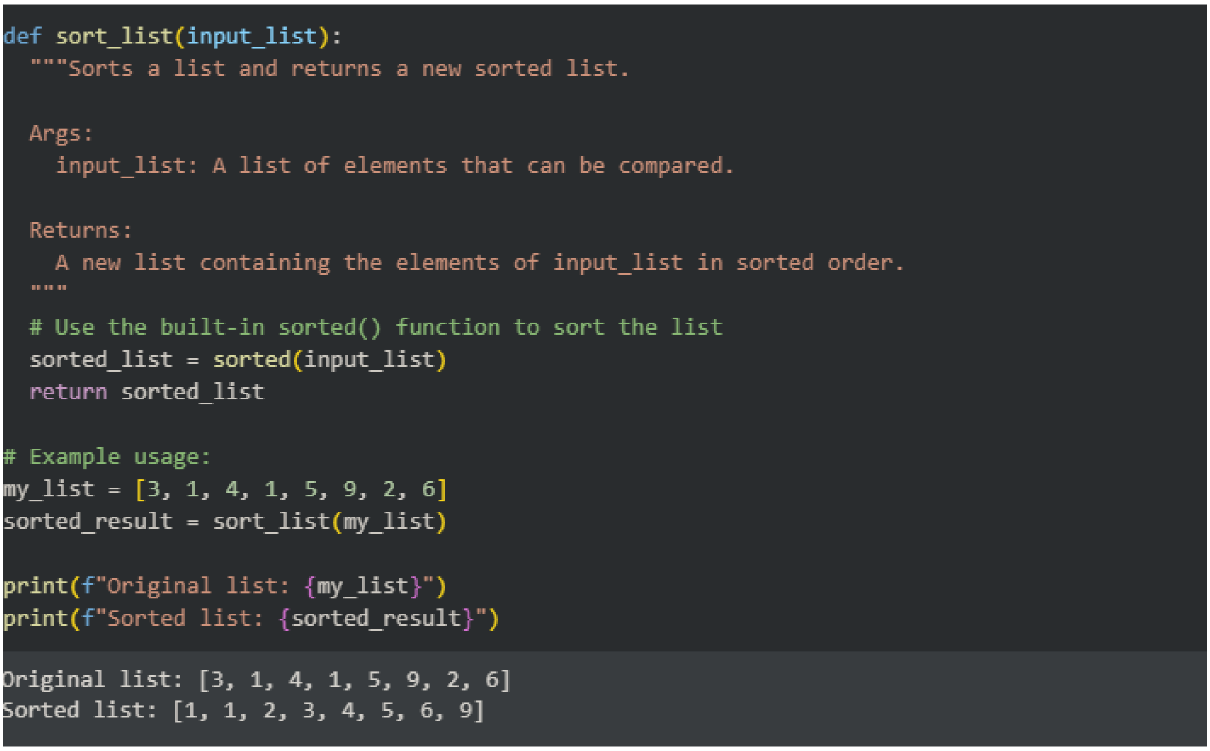
Provide a clear example input-output prompt to generate a sorting function

PROMPT:
generate a code for sorting in function where input and output is given clearly

EXPECTED OUTPUT:

Functional sorting code from AI

CODE WITH OUTPUT:



```
def sort_list(input_list):
  """Sorts a list and returns a new sorted list.

  Args:
    input_list: A list of elements that can be compared.

  Returns:
    A new list containing the elements of input_list in sorted order.
  """
  # Use the built-in sorted() function to sort the list
  sorted_list = sorted(input_list)
  return sorted_list

# Example usage:
my_list = [3, 1, 4, 1, 5, 9, 2, 6]
sorted_result = sort_list(my_list)

print(f"Original list: {my_list}")
print(f"Sorted list: {sorted_result}")
```

```
Original list: [3, 1, 4, 1, 5, 9, 2, 6]
Sorted list: [1, 1, 2, 3, 4, 5, 6, 9]
```

OBSERVATION:

The code defines a function called sort_list that takes a list of items (like numbers in our example) and sorts them from smallest to largest. It uses a built-in Python tool called sorted() to do this easily. The important thing is that it gives you a *new* list that is sorted, leaving the original list unchanged.

TASK DESCRIPTION 3:

QUESTION:

Start with the vague prompt “Generate python code to calculate power bill” and improve it step-by-step

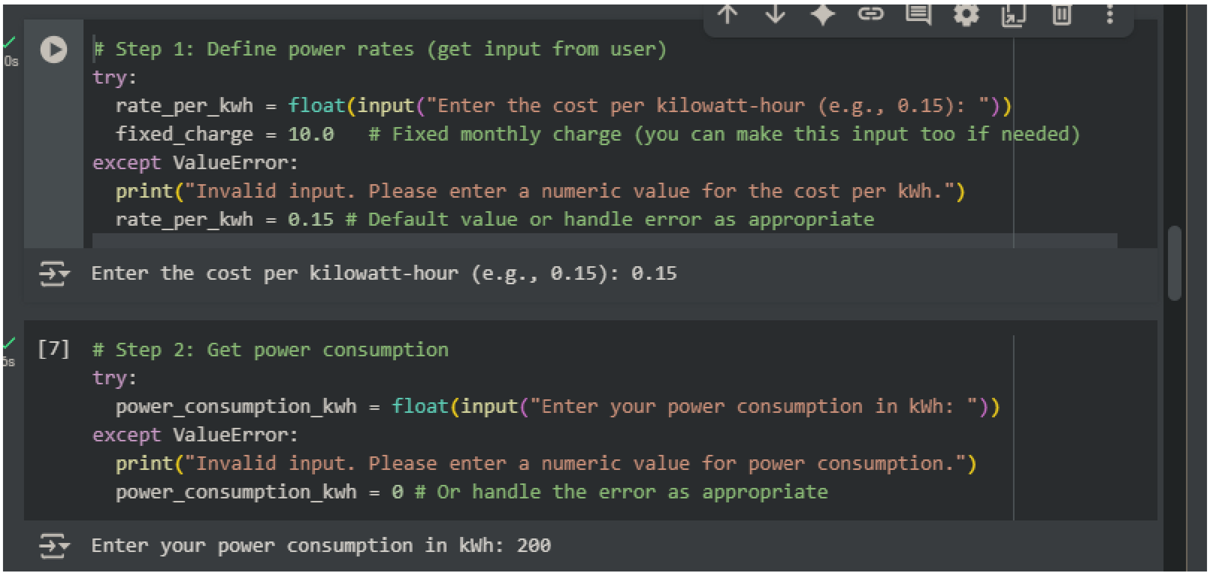
PROMPT:

Generate python code to calculate power bill by define power rates, get power consumption, calculate bill, display bill and finish the task.

EXPECTED OUTPUT:

Enhanced AI output with clearer prompts

CODE WITH OUTPUT:

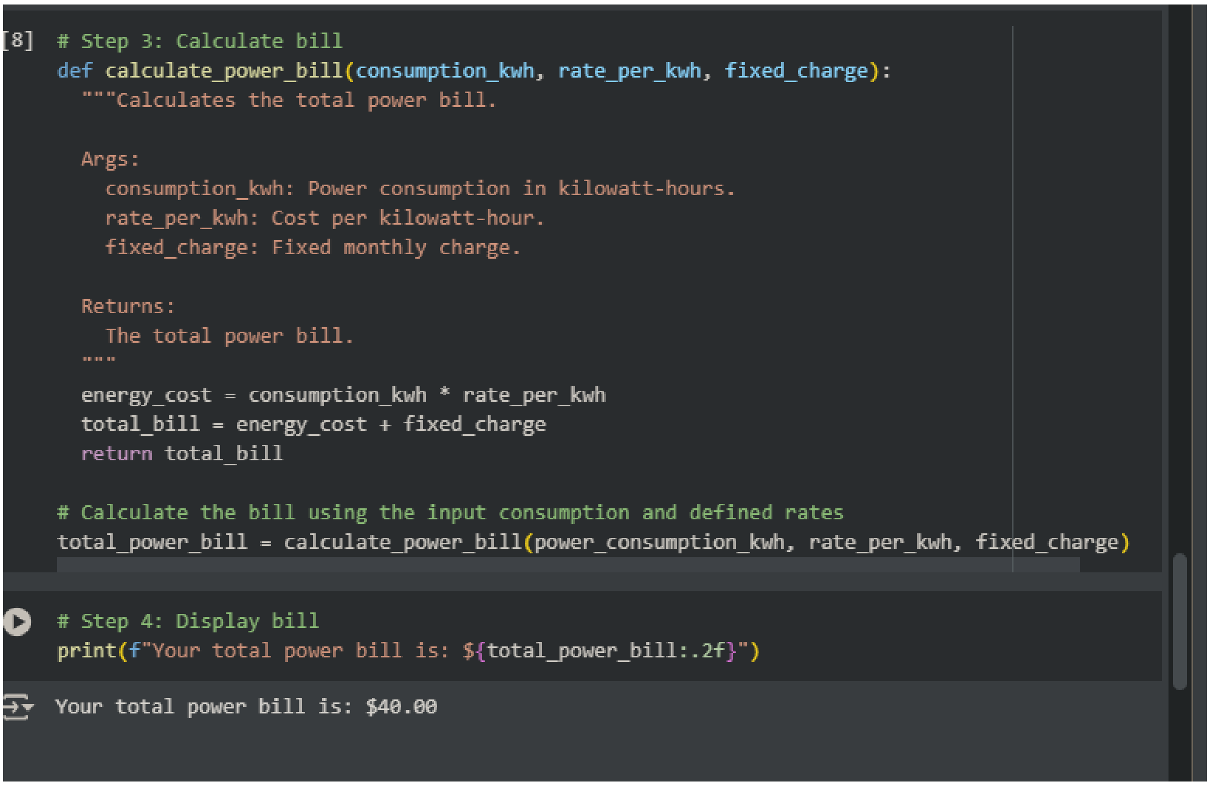


```
# Step 1: Define power rates (get input from user)
try:
  rate_per_kwh = float(input("Enter the cost per kilowatt-hour (e.g., 0.15): "))
  fixed_charge = 10.0   # Fixed monthly charge (you can make this input too if needed)
except ValueError:
  print("Invalid input. Please enter a numeric value for the cost per kWh.")
  rate_per_kwh = 0.15 # Default value or handle error as appropriate
```

```
Enter the cost per kilowatt-hour (e.g., 0.15): 0.15
```

```
# Step 2: Get power consumption
try:
  power_consumption_kwh = float(input("Enter your power consumption in kWh: "))
except ValueError:
  print("Invalid input. Please enter a numeric value for power consumption.")
  power_consumption_kwh = 0 # Or handle the error as appropriate
```

```
Enter your power consumption in kWh: 200
```



```
# Step 3: Calculate bill
def calculate_power_bill(consumption_kwh, rate_per_kwh, fixed_charge):
  """Calculates the total power bill.

  Args:
    consumption_kwh: Power consumption in kilowatt-hours.
    rate_per_kwh: Cost per kilowatt-hour.
    fixed_charge: Fixed monthly charge.

  Returns:
    The total power bill.
  """
  energy_cost = consumption_kwh * rate_per_kwh
  total_bill = energy_cost + fixed_charge
  return total_bill

# Calculate the bill using the input consumption and defined rates
total_power_bill = calculate_power_bill(power_consumption_kwh, rate_per_kwh, fixed_charge)
```

```
# Step 4: Display bill
print(f"Your total power bill is: ${total_power_bill:.2f}")
```

```
Your total power bill is: $40.00
```

OBSERVATION:

Based on the power consumption I entered (200 kWh) and the example rates we used ($0.15 per kWh plus a $$0.15 per kWh plus a $10.00 fixed charge), the code correctly calculated and showed that the total power bill is $40.00. This means the energy cost was $$40.00. This means the energy cost was $30.00 (200 * $0.15) plus the $$0.15) plus the $10.00 fixed charge.

TASK DESCRIPTION 4:

QUESTION:

Write structured comments to help AI generate two linked functions (e.g., login_user() and register_user()).

PROMPT:

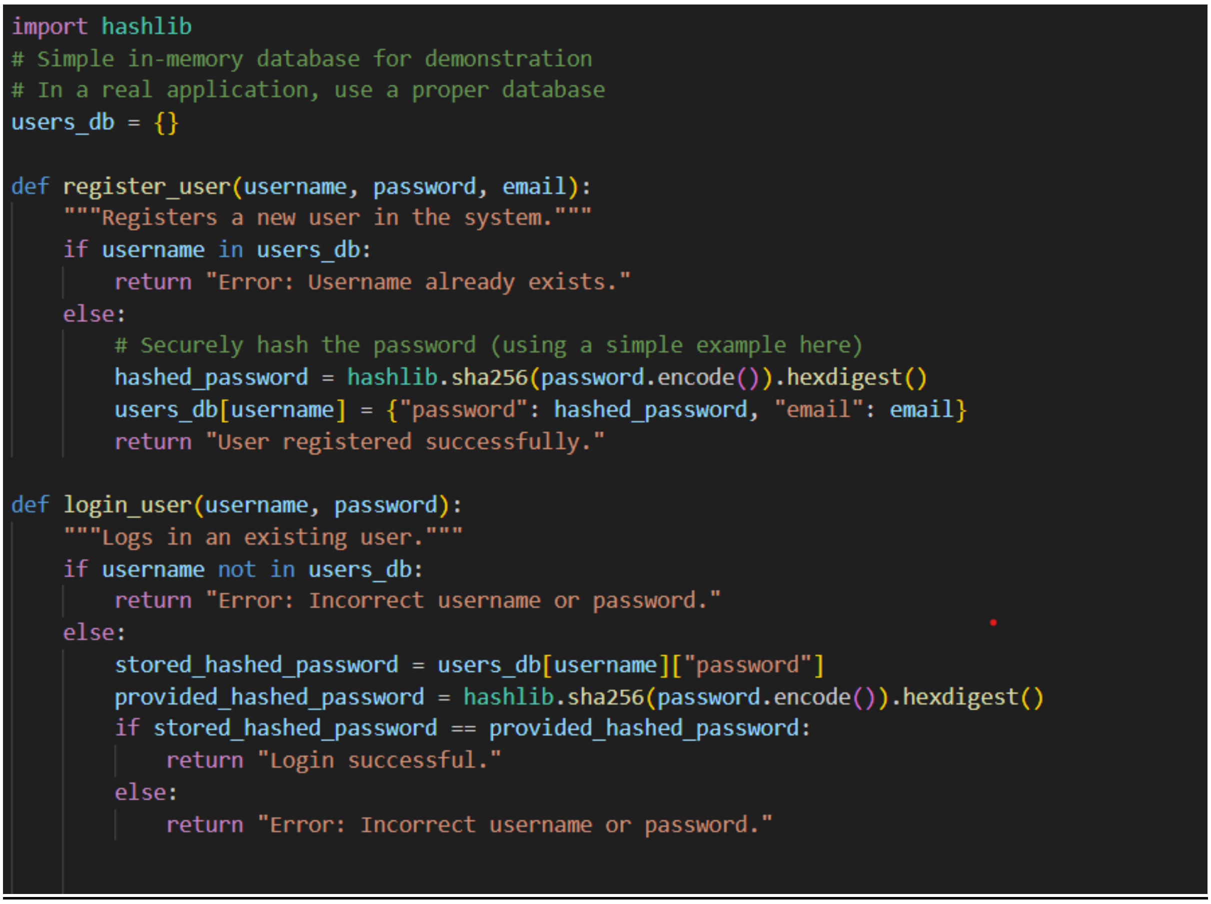
# Function: register_user # Goal: Register a new user in the system. # Inputs: # - username: The desired username (string). # - password: The user's password (string). # - email: The user's email address (string). # Outputs: # - Success message if registration is successful (string). # - Error message if registration fails (e.g., username already exists) (string). # Steps: # 1. Check if the username already exists in the database. # 2. If username exists, return an error message. # 3. If username does not exist, securely hash the password. # 4. Store the

username, hashed password, and email in the database. # 5. Return a success message. # Linked Function: This function is linked to login_user(). A user needs to register before they can log in.

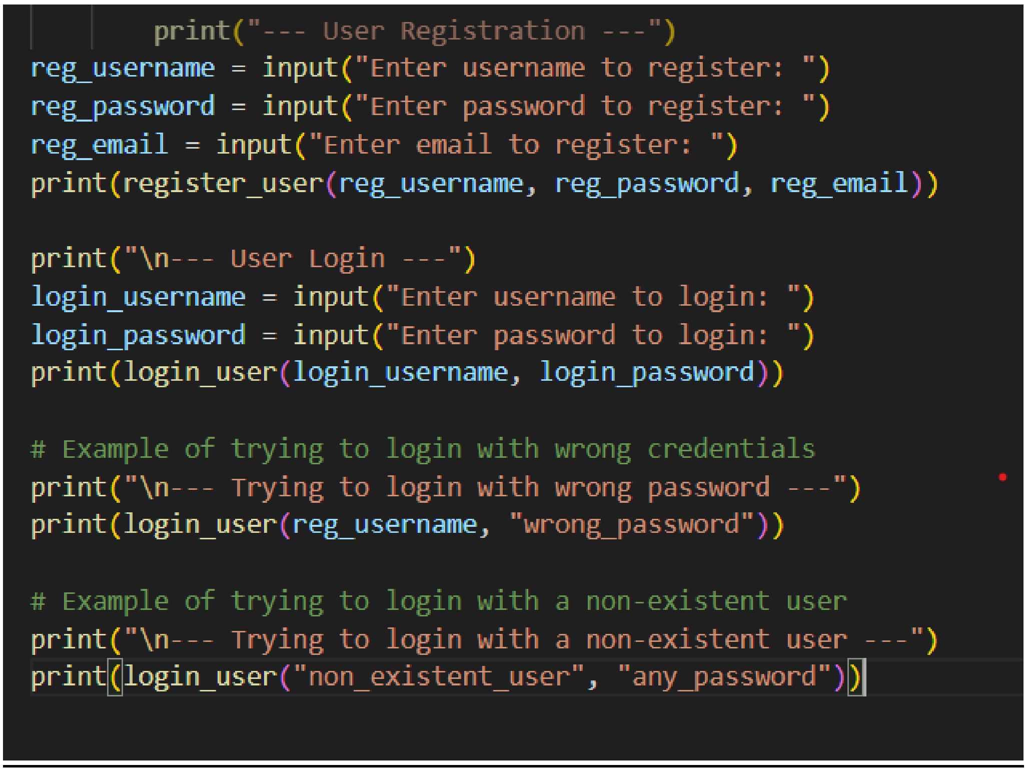
EXPECTED OUTPUT:

Consistent functions with shared logic

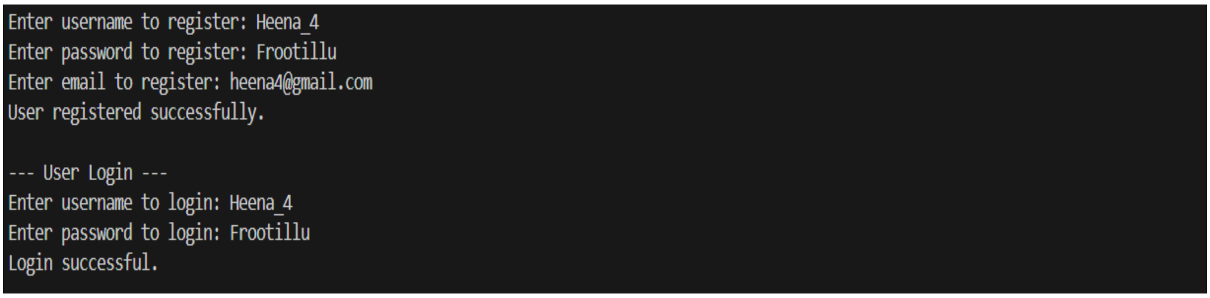
CODE WITH OUTPUT:



```python
import hashlib
# Simple in-memory database for demonstration
# In a real application, use a proper database
users_db = {}

def register_user(username, password, email):
    """Registers a new user in the system."""
    if username in users_db:
        return "Error: Username already exists."
    else:
        # Securely hash the password (using a simple example here)
        hashed_password = hashlib.sha256(password.encode()).hexdigest()
        users_db[username] = {"password": hashed_password, "email": email}
        return "User registered successfully."

def login_user(username, password):
    """Logs in an existing user."""
    if username not in users_db:
        return "Error: Incorrect username or password."
    else:
        stored_hashed_password = users_db[username]["password"]
        provided_hashed_password = hashlib.sha256(password.encode()).hexdigest()
        if stored_hashed_password == provided_hashed_password:
            return "Login successful."
        else:
            return "Error: Incorrect username or password."
```



```
        print("--- User Registration ---")
reg_username = input("Enter username to register: ")
reg_password = input("Enter password to register: ")
reg_email = input("Enter email to register: ")
print(register_user(reg_username, reg_password, reg_email))

print("\n--- User Login ---")
login_username = input("Enter username to login: ")
login_password = input("Enter password to login: ")
print(login_user(login_username, login_password))

# Example of trying to login with wrong credentials
print("\n--- Trying to login with wrong password ---")
print(login_user(reg_username, "wrong_password"))

# Example of trying to login with a non-existent user
print("\n--- Trying to login with a non-existent user ---")
print(login_user("non_existent_user", "any_password"))
```



```
Enter username to register: Heena_4
Enter password to register: Frootillu
Enter email to register: heena4@gmail.com
User registered successfully.

--- User Login ---
Enter username to login: Heena_4
Enter password to login: Frootillu
Login successful.
```

OBSERVATION:

The main observation is that the code correctly distinguishes between valid and invalid login attempts.

- When you provide the **correct username and password** that you registered, the code outputs a "Login successful" message.
- When you provide an **incorrect password** for a registered user, or try to log in with a **username that doesn't exist**, the code correctly outputs an "Error: Incorrect username or password" message.

TASK DESCRIPTION 5:

QUESTION:

Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

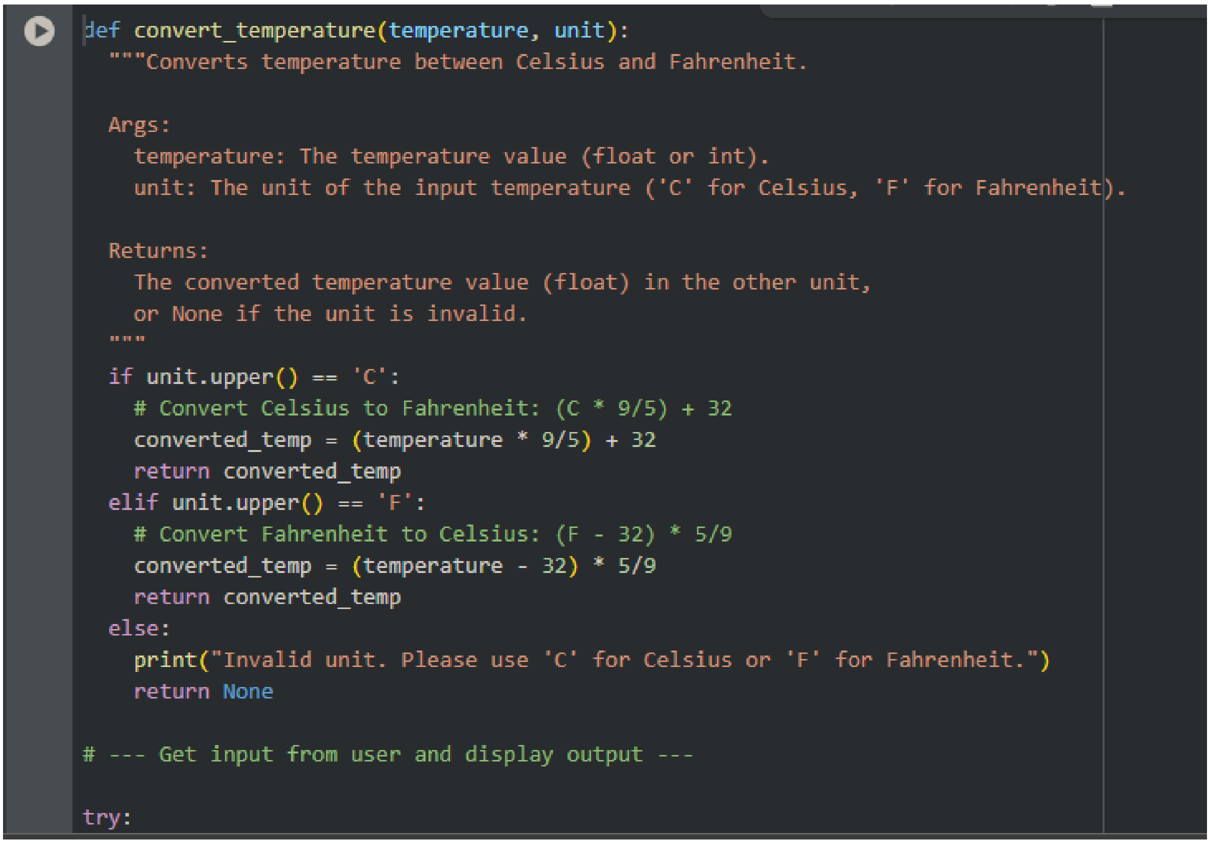
PROMPT1:

Generate a code Temperature Conversion Function with Clear Instructions with user's input and output.

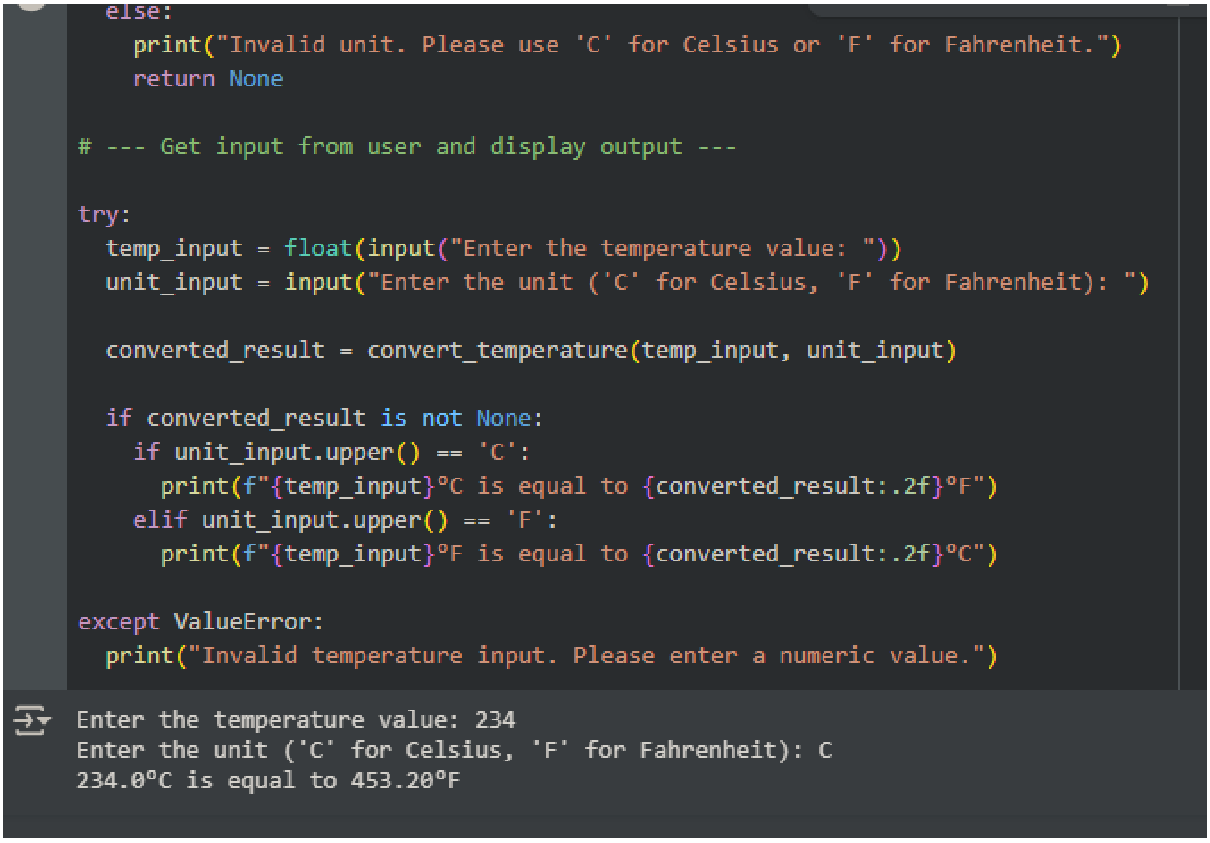
EXPECTED OUTPUT:

Code quality difference analysis for various prompts

CODE WITH OUTPUT:



```
def convert_temperature(temperature, unit):
  """Converts temperature between Celsius and Fahrenheit.

  Args:
    temperature: The temperature value (float or int).
    unit: The unit of the input temperature ('C' for Celsius, 'F' for Fahrenheit).

  Returns:
    The converted temperature value (float) in the other unit,
    or None if the unit is invalid.
  """
  if unit.upper() == 'C':
    # Convert Celsius to Fahrenheit: (C * 9/5) + 32
    converted_temp = (temperature * 9/5) + 32
    return converted_temp
  elif unit.upper() == 'F':
    # Convert Fahrenheit to Celsius: (F - 32) * 5/9
    converted_temp = (temperature - 32) * 5/9
    return converted_temp
  else:
    print("Invalid unit. Please use 'C' for Celsius or 'F' for Fahrenheit.")
    return None

# --- Get input from user and display output ---

try:
```



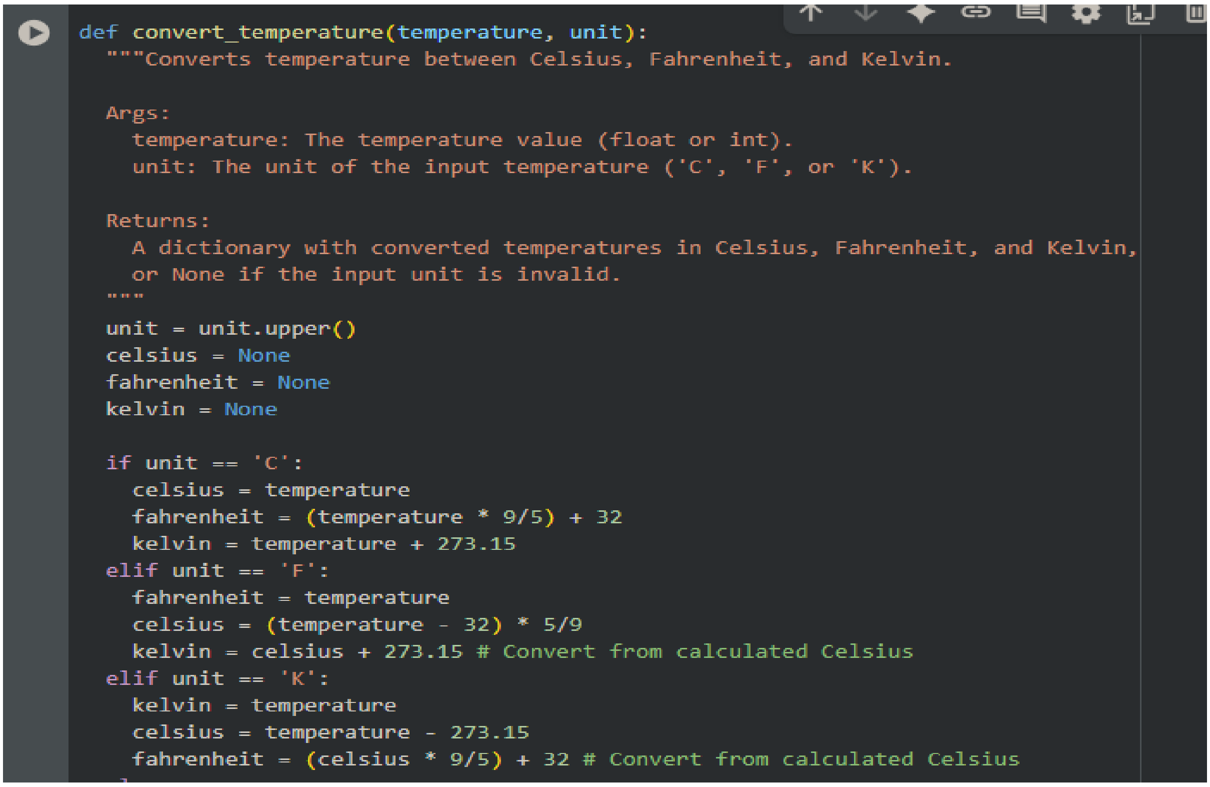
```
  else:
    print("Invalid unit. Please use 'C' for Celsius or 'F' for Fahrenheit.")
    return None

# --- Get input from user and display output ---

try:
  temp_input = float(input("Enter the temperature value: "))
  unit_input = input("Enter the unit ('C' for Celsius, 'F' for Fahrenheit): ")

  converted_result = convert_temperature(temp_input, unit_input)

  if converted_result is not None:
    if unit_input.upper() == 'C':
      print(f"{temp_input}°C is equal to {converted_result:.2f}°F")
    elif unit_input.upper() == 'F':
      print(f"{temp_input}°F is equal to {converted_result:.2f}°C")

except ValueError:
  print("Invalid temperature input. Please enter a numeric value.")
```

```
Enter the temperature value: 234
Enter the unit ('C' for Celsius, 'F' for Fahrenheit): C
234.0°C is equal to 453.20°F
```

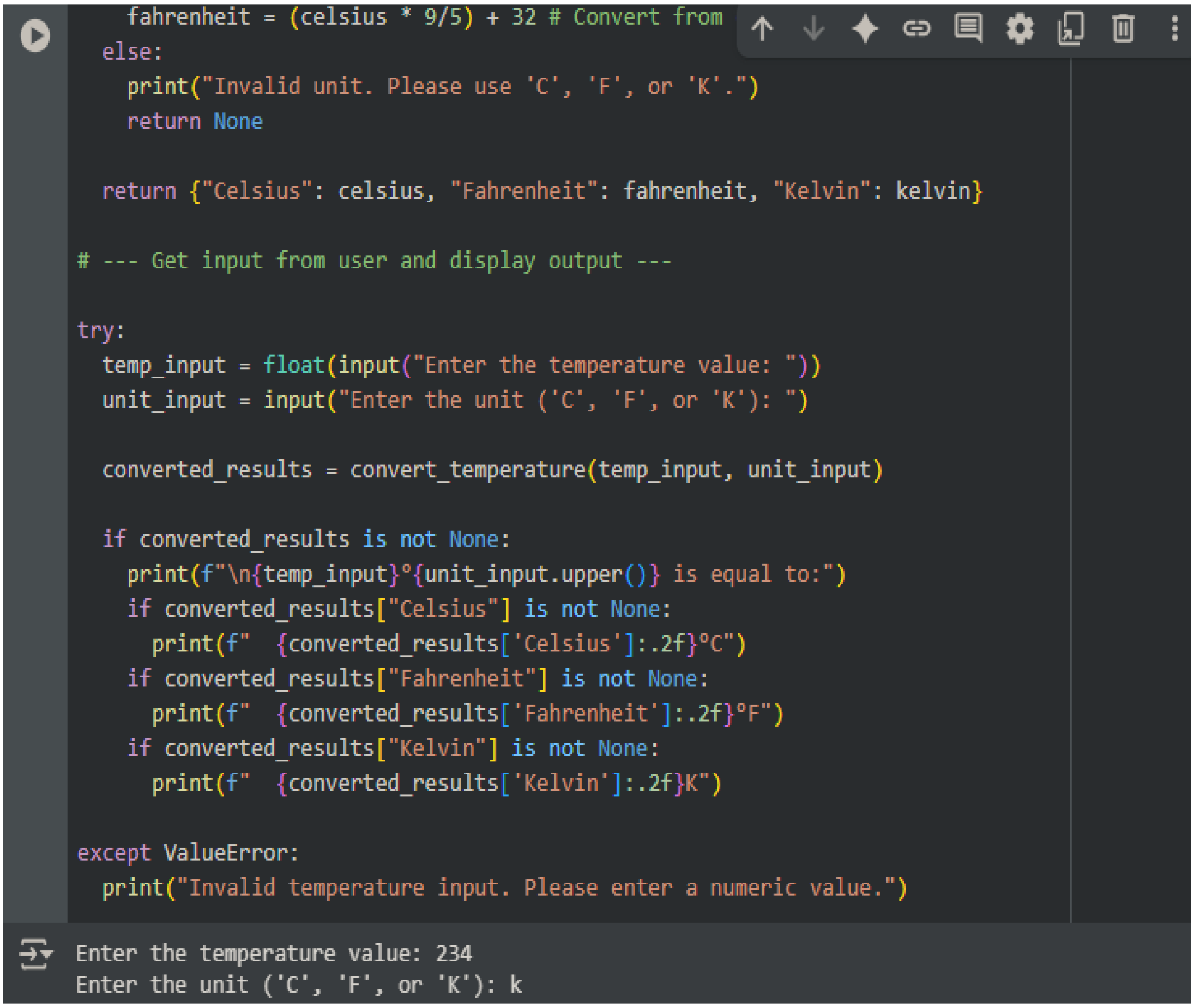
PROMPT 2:

Generate a code Temperature Conversion Function with kelvin,celsiun,fahrenheit.

CODE WITH OUTPUT:

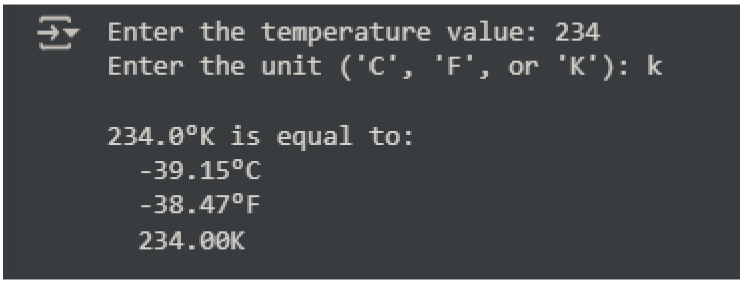


```
def convert_temperature(temperature, unit):
  """Converts temperature between Celsius, Fahrenheit, and Kelvin.

  Args:
    temperature: The temperature value (float or int).
    unit: The unit of the input temperature ('C', 'F', or 'K').

  Returns:
    A dictionary with converted temperatures in Celsius, Fahrenheit, and Kelvin,
    or None if the input unit is invalid.
  """
  unit = unit.upper()
  celsius = None
  fahrenheit = None
  kelvin = None

  if unit == 'C':
    celsius = temperature
    fahrenheit = (temperature * 9/5) + 32
    kelvin = temperature + 273.15
  elif unit == 'F':
    fahrenheit = temperature
    celsius = (temperature - 32) * 5/9
    kelvin = celsius + 273.15 # Convert from calculated Celsius
  elif unit == 'K':
    kelvin = temperature
    celsius = temperature - 273.15
    fahrenheit = (celsius * 9/5) + 32 # Convert from calculated Celsius
```



```
    fahrenheit = (celsius * 9/5) + 32 # Convert from
  else:
    print("Invalid unit. Please use 'C', 'F', or 'K'.")
    return None

  return {"Celsius": celsius, "Fahrenheit": fahrenheit, "Kelvin": kelvin}

# --- Get input from user and display output ---

try:
  temp_input = float(input("Enter the temperature value: "))
  unit_input = input("Enter the unit ('C', 'F', or 'K'): ")

  converted_results = convert_temperature(temp_input, unit_input)

  if converted_results is not None:
    print(f"\n{temp_input}°{unit_input.upper()} is equal to:")
    if converted_results["Celsius"] is not None:
      print(f"  {converted_results['Celsius']:.2f}°C")
    if converted_results["Fahrenheit"] is not None:
      print(f"  {converted_results['Fahrenheit']:.2f}°F")
    if converted_results["Kelvin"] is not None:
      print(f"  {converted_results['Kelvin']:.2f}K")

except ValueError:
  print("Invalid temperature input. Please enter a numeric value.")
```

```
Enter the temperature value: 234
Enter the unit ('C', 'F', or 'K'): k
```



```
Enter the temperature value: 234
Enter the unit ('C', 'F', or 'K'): k

234.0°K is equal to:
  -39.15°C
  -38.47°F
  234.00K
```

OBSERVATION:

The key difference is the **scope of conversion** and the **output format**. The first version did a single conversion and returned one value, while the second version does multiple conversions and returns a dictionary of results, providing a more complete view of the temperature across different scales. The second version is more versatile as it includes Kelvin, which is important in scientific contexts.