

# **AI ASSISTED CODING**

## **ASSIGNMENT 3.3**

**PROGRAM: B.TECH/CSE**

**NAME : NAREDDY JASMITHA**

**ENROLLMENT NO: 2403A51281**

**COURSE :AI ASSISTED CODE**

**BATCH:01**

**DATE: 20-08-2025**

### **#TASK DESCRIPTION 1:**

Try 3 different prompts to generate a factorial function

### **PROMPT:**

Develop a code to generate 3 different types of prompts of a factorial function

### **EXPECTED OUTPUT 1:**

Comparison of AI-generated code styles

```
import math

# Implementation 1: Iterative approach
def factorial_iterative(n):
    """Calculates the factorial of a non-negative integer using an iterative approach."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
    result = 1
    for i in range(1, n + 1):
        result *= i
    return result

# Implementation 2: Recursive approach
def factorial_recursive(n):
    """Calculates the factorial of a non-negative integer using a recursive approach."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Implementation 3: Using the math module
def factorial_math(n):
    """Calculates the factorial of a non-negative integer using the math module."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
```

```
# Implementation 2: Recursive approach
def factorial_recursive(n):
    """Calculates the factorial of a non-negative integer using a recursive approach."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
    if n == 0:
        return 1
    else:
        return n * factorial_recursive(n - 1)

# Implementation 3: Using the math module
def factorial_math(n):
    """Calculates the factorial of a non-negative integer using the math module."""
    if n < 0:
        return "Factorial is not defined for negative numbers"
    return math.factorial(n)

# Example usage:
num = 5

print(f"Iterative factorial of {num}: {factorial_iterative(num)}")
print(f"Recursive factorial of {num}: {factorial_recursive(num)}")
print(f"Math module factorial of {num}: {factorial_math(num)}")
```

```
Iterative factorial of 5: 120
Recursive factorial of 5: 120
Math module factorial of 5: 120
```

## EXPLANATION:

---

non-negative integer in Python.

Here's a breakdown of each implementation:

1. **factorial\_iterative(n)**: This function calculates the factorial using a loop. It initializes a `result` to 1 and then iterates from 1 up to `n`, multiplying `result` by each number in the range. This is a straightforward and often more memory-efficient approach for larger numbers compared to recursion.
2. **factorial\_recursive(n)**: This function uses recursion to calculate the factorial. It defines a base case where the factorial of 0 is 1. For any other positive number `n`, it returns `n` multiplied by the factorial of `n-1`. This approach is often considered more elegant for problems that can be broken down into smaller, self-similar subproblems, but it can lead to stack overflow errors for very large inputs due to deep recursion.
3. **factorial\_math(n)**: This function leverages the built-in `math.factorial()` function from Python's `math` module. This is the most concise and generally the most efficient way to calculate factorials for non-negative integers in Python, as it's implemented in optimized C code.

## #TASK DESCRIPTION 2:

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

### PROMPT:

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

### EXPECTED OUTPUT:

Functional sorting code from AI

```
def generate_sorting_prompt(input_list, sorted_list):
    """Generates an input-output prompt for a sorting function.

    Args:
        input_list: The example unsorted input list.
        sorted_list: The expected sorted output list.

    Returns:
        A string containing the input-output prompt.
    """
    prompt = f"Write a Python function that sorts a list of numbers.\n\n"
    prompt += f"Input:\n{input_list}\n\n"
    prompt += f"Output:\n{sorted_list}"
    return prompt

# Example usage:
example_input = [3, 1, 4, 1, 5, 9, 2, 6]
example_output = sorted(example_input) # Use Python's built-in sorted function for the expected output

sorting_prompt = generate_sorting_prompt(example_input, example_output)
print(sorting_prompt)
```

Write a Python function that sorts a list of numbers.

Input:  
[3, 1, 4, 1, 5, 9, 2, 6]

Output:  
[1, 1, 2, 3, 4, 5, 6, 9]

## EXPLANATION:

1. **Function Definition:** The code defines a function `generate_sorting_prompt` that takes two arguments: `input_list` (the unsorted list) and `sorted_list` (the expected sorted list).
2. **Prompt Construction:** Inside the function, it builds a string variable called `prompt`.
  - It starts with a general instruction: "Write a Python function that sorts a list of numbers."
  - It then adds the `Input:` label followed by the string representation of the `input_list`.
  - Finally, it adds the `Output:` label followed by the string representation of the `sorted_list`.
3. **Return Value:** The function returns the complete `prompt` string.
4. **Example Usage:**
  - An example unsorted list `example_input` is created.
  - The `sorted()` built-in Python function is used to easily generate the expected sorted list `example_output` from `example_input`.
  - The `generate_sorting_prompt` function is called with `example_input` and `example_output` to create the `sorting prompt`.

## #TASK DESCRIPTION 3:

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

## PROMPT:

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

## EXPECTED OUTPUT:

Enhanced AI output with clearer prompt

```
# The vague prompt: "Generate python code to calculate power bill"

# Essential information missing to calculate a power bill:
missing_info = [
    "Electricity consumption (in kwh) for the billing period.",
    "Electricity rates (e.g., per kwh, tiered rates, time-of-use rates).",
    "Fixed charges (e.g., service fees, meter charges).",
    "Taxes and surcharges (e.g., local taxes, state taxes, environmental fees).",
    "Any applicable discounts or credits."
]

print("Missing information to calculate a power bill:")
for item in missing_info:
    print(f"- {item}")
```

Missing information to calculate a power bill:

- Electricity consumption (in kwh) for the billing period.
- Electricity rates (e.g., per kwh, tiered rates, time-of-use rates).
- Fixed charges (e.g., service fees, meter charges).
- Taxes and surcharges (e.g., local taxes, state taxes, environmental fees).
- Any applicable discounts or credits.

```
vague_prompt = "Generate python code to calculate power bill"
prompt_with_consumption = vague_prompt + " by asking the user for their electricity consumption in kilowatt-
print(prompt_with_consumption)
```

Generate python code to calculate power bill by asking the user for their electricity consumption in kilowatt

🔍 # Refined prompt: Generate python code to calculate power bill by asking the user for their electricity consump

```
def calculate_power_bill():  
    """Calculates the power bill based on consumption, rate, and additional charges."""  
  
    # 1. Prompt the user for electricity consumption  
    try:  
        consumption_kwh = float(input("Enter your electricity consumption in kwh: "))  
    except ValueError:  
        print("Invalid input. Please enter a numerical value for consumption.")  
        return  
  
    # 3. Define a single electricity rate per kwh (placeholder)  
    rate_per_kwh = 0.15 # Example rate in dollars per kwh  
  
    # 4. Include variables for additional charges (placeholders)  
    tax_rate = 0.05 # Example tax rate (5%)  
    service_fee = 2.50 # Example fixed service fee  
  
    # 5. Calculate the total power bill  
    cost_based_on_consumption = consumption_kwh * rate_per_kwh  
    taxes = cost_based_on_consumption * tax_rate  
    total_bill = cost_based_on_consumption + taxes + service_fee  
  
    # 6. Print the calculated total power bill  
    print(f"\n--- Power Bill Summary ---")  
    print(f"Consumption: {consumption_kwh:.2f} kwh")  
    print(f"Rate: ${rate_per_kwh:.2f} per kwh")  
    print(f"Cost based on consumption: ${cost_based_on_consumption:.2f}")
```

✓ 6m [13]

```
    # 4. Include variables for additional charges (placeholders)  
    tax_rate = 0.05 # Example tax rate (5%)  
    service_fee = 2.50 # Example fixed service fee  
  
    # 5. Calculate the total power bill  
    cost_based_on_consumption = consumption_kwh * rate_per_kwh  
    taxes = cost_based_on_consumption * tax_rate  
    total_bill = cost_based_on_consumption + taxes + service_fee  
  
    # 6. Print the calculated total power bill  
    print(f"\n--- Power Bill Summary ---")  
    print(f"Consumption: {consumption_kwh:.2f} kwh")  
    print(f"Rate: ${rate_per_kwh:.2f} per kwh")  
    print(f"Cost based on consumption: ${cost_based_on_consumption:.2f}")  
    print(f"Taxes ({tax_rate*100}%): ${taxes:.2f}")  
    print(f"Service Fee: ${service_fee:.2f}")  
    print(f"-----")  
    print(f"Total Power Bill: ${total_bill:.2f}")  
  
# Execute the function  
calculate_power_bill()
```

➡ Enter your electricity consumption in kwh: 200

```
--- Power Bill Summary ---  
Consumption: 200.00 kwh  
Rate: $0.15 per kwh  
Cost based on consumption: $30.00  
Taxes (5.0%): $1.50  
Service Fee: $2.50  
-----  
Total Power Bill: $34.00
```

## EXPLANATION:

1. **Function Definition:** It defines a function `calculate_power_bill()` that encapsulates the entire logic for calculating and summarizing the power bill.
2. **Get Consumption Input:** It prompts the user to enter their electricity consumption in kilowatt-hours (kWh) using the `input()` function. It uses a `try-except` block to handle potential `ValueError` if the user enters non-numeric input.
3. **Define Rate and Charges:** It defines placeholder variables for:
  - `rate_per_kwh`: An example electricity rate per kWh (set to 0.15).
  - `tax_rate`: An example tax rate (set to 0.05 for 5%).
  - `service_fee`: An example fixed service fee (set to 2.50).
  - *Note: These are hardcoded values for demonstration, and in a real application, these would likely come from a configuration or user input.*
4. **Calculate Costs:**
  - `cost_based_on_consumption`: Calculates the cost based on the entered consumption and the `rate_per_kwh`.
  - `taxes`: Calculates the tax amount by applying the `tax_rate` to the `cost_based_on_consumption`.



- `tax_rate` : An example tax rate (set to 0.05 for 5%).
- `service_fee` : An example fixed service fee (set to 2.50).
- *Note: These are hardcoded values for demonstration, and in a real application, these would likely come from a configuration or user input.*

#### 4. Calculate Costs:

- `cost_based_on_consumption` : Calculates the cost based on the entered consumption and the `rate_per_kwh`.
- `taxes` : Calculates the tax amount by applying the `tax_rate` to the `cost_based_on_consumption`.
- `total_bill` : Calculates the final bill by summing the `cost_based_on_consumption`, `taxes`, and the `service_fee`.

5. **Print Summary:** It prints a formatted summary of the power bill, including the consumption, rate, cost based on consumption, taxes, service fee, and the final total bill.

6. **Execute Function:** The `calculate_power_bill()` function is called at the end to run the code and perform the calculation when the cell is executed.

## #Task Description#4

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

### PROMPT:

Develop a code to generate ask Description#4 • Write structured comments to help AI generate two linked functions (e.g., `login_user()` and `register_user()`)

### EXPECTED OUTPUT:

## Consistent functions with shared logic

```
# Function 1: Register User
# This function should handle the process of creating a new user account.
# It should take parameters like username, password, and email.
# It should perform validation (e.g., check if username already exists).
# It should securely store the user's credentials (e.g., hash the password).
# It should return a status indicating success or failure of the registration.
def register_user(username, password, email):
    """Registers a new user with the provided username, password, and email."""
    # TODO: Implement user registration logic
    pass

# Function 2: Login User
# This function should handle the process of authenticating an existing user.
# It should take parameters like username and password.
# It should verify the provided credentials against stored user data.
# It should return a status indicating successful login or authentication failure.
# This function is linked to the register_user function as it operates on the user data created by it.
def login_user(username, password):
    """Logs in an existing user by verifying the provided username and password."""
    # TODO: Implement user login logic
    pass

# Example usage (optional - for demonstration after implementation)
# if register_user("test_user", "secure_password", "test@example.com"):
#     print("User registered successfully.")
# if login_user("test_user", "secure_password"):
#     print("User logged in successfully.")
# else:
#     print("Login failed.")
# else:
#     print("User registration failed.")
```

## EXPLANATION:

This code cell is a template for creating user authentication functions in Python. It doesn't contain the full implementation yet, but it uses comments and function definitions to describe what two key functions, `register_user` and `login_user`, should do.

- `register_user(username, password, email)`: This function is intended for creating new user accounts. The comments explain that it should handle validation (like checking if a username is already taken), securely store the password (you'd typically hash it), and indicate whether the registration was successful.
- `login_user(username, password)`: This function is for authenticating existing users. The comments specify that it should verify the provided username and password against stored data. Importantly, it mentions that this function is *linked* to `register_user` because it operates on the user data created by the registration process.

## #TASK DESCRIPTION 5:

Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions

### PROMPT:

generate a code on temperature conversion function

### EXPECTED OUTPUT:

Code quality difference analysis for various prompt

```
def convert_temperature_refined(temp, input_unit, output_unit):
    """Converts temperature between Celsius, Fahrenheit, and Kelvin.

    Args:
        temp: The temperature value (numerical).
        input_unit: The unit of the input temperature ('C', 'F', or 'K').
        output_unit: The desired unit for the output temperature ('C', 'F', or 'K').

    Returns:
        The converted temperature as a float, or None for invalid input.
    """
    # Check for invalid temperature input type
    if not isinstance(temp, (int, float)):
        return None

    # Convert input unit to uppercase for case-insensitive comparison
    input_unit = input_unit.upper()
    output_unit = output_unit.upper()

    # Check for invalid unit strings
    supported_units = ['C', 'F', 'K']
    if input_unit not in supported_units or output_unit not in supported_units:
        return None

    # Convert input temperature to Kelvin (base unit)
    if input_unit == 'C':
        temp_k = temp + 273.15
    elif input_unit == 'F':
        temp_k = (temp - 32) * 5/9 + 273.15
    elif input_unit == 'K':
```

```
if input_unit == 'C':
    temp_k = temp + 273.15
elif input_unit == 'F':
    temp_k = (temp - 32) * 5/9 + 273.15
elif input_unit == 'K':
    temp_k = temp

# Convert from Kelvin to the desired output unit
if output_unit == 'C':
    converted_temp = temp_k - 273.15
elif output_unit == 'F':
    converted_temp = (temp_k - 273.15) * 9/5 + 32
elif output_unit == 'K':
    converted_temp = temp_k

return converted_temp

# Example usage:
print(f"25°C to Fahrenheit: {convert_temperature_refined(25, 'C', 'F'):.2f}°F")
print(f"68°F to Celsius: {convert_temperature_refined(68, 'F', 'C'):.2f}°C")
print(f"273.15K to Celsius: {convert_temperature_refined(273.15, 'K', 'C'):.2f}°C")
print(f"0°C to Kelvin: {convert_temperature_refined(0, 'C', 'K'):.2f}K")
print(f"Invalid input (string temp): {convert_temperature_refined('abc', 'C', 'F')}")
print(f"Invalid input (unsupported unit): {convert_temperature_refined(25, 'X', 'F')}")
```

25°C to Fahrenheit: 77.00°F  
68°F to Celsius: 20.00°C  
273.15K to Celsius: 0.00°C  
0°C to Kelvin: 273.15K  
Invalid input (string temp): None  
Invalid input (unsupported unit): None

## EXPLANATION:

1. **Function Signature:** The function `convert_temperature_refined` takes three arguments:

- `temp`: The numerical value of the temperature you want to convert.
- `input_unit`: A string representing the current unit of the temperature ('C', 'F', or 'K').
- `output_unit`: A string representing the unit you want to convert the temperature to ('C', 'F', or 'K').

2. **Input Validation:**

- It first checks if the `temp` input is a number (integer or float) using `isinstance`. If not, it returns `None` to indicate invalid input.
- It converts the `input_unit` and `output_unit` strings to uppercase using `.upper()` to make the unit checking case-insensitive (so 'c', 'f', 'k' work as well).
- It checks if both the `input_unit` and `output_unit` are among the supported units ('C', 'F', or 'K'). If either is invalid, it returns `None`.