

ASSIGNMENT-3.3

Name: D. Sravika Reddy

Hall-Ticket No: 2403a510d0

Batch No: 05

Course: AI Assisted Coding

Task Description#1

- Try 3 different prompts to generate a factorial function.

Expected Output#1

- Comparison of AI-generated code styles

Prompt 1 – Zero-Shot Prompting

Prompt:

"Write a Python function to calculate the factorial of a number."

Code

python

```
def factorial_zero_shot(n):  
    result = 1  
    for i in range(1, n + 1):  
        result *= i  
    return result
```

- **Style:** Minimal and direct, no error handling.
- **Limitation:** Does not handle negative numbers or special cases like $0!$.

Prompt 2 – One-Shot Prompting

Prompt:

"Write a Python function to calculate factorial. Example: factorial(5) → 120."

Code

python

```
def factorial_one_shot(n):  
    if n < 0:  
        return "Factorial not defined for negative numbers"  
    elif n == 0:  
        return 1  
    else:  
        result = 1  
        for i in range(1, n + 1):  
            result *= i  
        return result
```

Prompt 3 – Few-Shot Prompting

Prompt:
"Generate a factorial function.
Example 1: factorial(0) → 1
Example 2: factorial(5) → 120
Example 3: factorial(-3) → 'Factorial not defined for negative numbers'"

Code

```
python

def factorial_few_shot(n):
    if n < 0:
        return "Factorial not defined for negative numbers"
    elif n == 0:
        return 1
    else:
        return n * factorial_few_shot(n - 1) # Recursive style
```

Comparison of AI-Generated Code Styles

Prompt Style	Code Complexity	Error Handling	Approach
Zero-Shot	Simple, minimal	None	Iterative
One-Shot	Moderate	Yes (negative & zero)	Iterative
Few-Shot	Most refined	Yes, all cases	Recursive

Task Description#2

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

Expected Output#2

- Functional sorting code from AI

```
def sort_numbers(numbers):
    """
    Sorts a list of numbers in ascending order.
    """
    return sorted(numbers)

# Example Usage:
print(sort_numbers([5, 2, 9, 1])) # Output: [1, 2, 5, 9]
```

Output

```
csharp

[1, 2, 5, 9]
```

Explanation

- Uses Python's built-in `sorted()` function for efficiency.
- One-shot prompt gave a **clear input-output pair**, ensuring correct behavior.

Task Description#3

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

improve it step-by-step

Expected Output#3

- Enhanced AI output with clearer prompts

1. Vague Prompt: "Generate python code to calculate power bill"

This prompt is too general. A bill calculation requires specific inputs like the **cost per unit**, the **number of units consumed**, and any **additional charges or taxes**. The code below makes assumptions, which isn't ideal.

Python

```
# Assuming a flat rate of $0.15 per unit and 300 units consumed
units_consumed = 300
rate_per_unit = 0.15

# Calculate the total bill
total_bill = units_consumed * rate_per_unit

# Print the result
print(f"Your power bill is: ${total_bill:.2f}")
```

2. Improved Prompt: "Generate python code to calculate power bill based on user input for units consumed and a fixed rate."

This is a better prompt. It specifies that the **units consumed** should be provided by the user, making the code more interactive and practical. However, the **rate** is still fixed, which is often not how a real bill works.

Python

```
# Get units consumed from the user
try:
    units_consumed = float(input("Enter the number of units consumed: "))
    # Define a fixed rate per unit
    rate_per_unit = 0.18 # Example rate: $0.18 per unit

    # Calculate the total bill
    total_bill = units_consumed * rate_per_unit

    # Print the result
    print(f"Your power bill for {units_consumed} units is: ${total_bill:.2f}")

except ValueError:
    print("Invalid input. Please enter a numerical value for units consumed.")
```

3. Advanced Prompt: "Generate python code to calculate a tiered power bill, including a fixed charge and GST. The bill should be calculated based on the following tiers:

- First 100 units at \$0.20 per unit.
- Next 200 units at \$0.25 per unit.
- Remaining units at \$0.30 per unit.
- A fixed monthly charge of \$15.
- A Goods and Services Tax (GST) of 18% on the total amount."

This is the most effective prompt. It provides a detailed, real-world scenario with specific **tiers**, **fixed charges**, and **taxes**. This level of detail allows the code to accurately model a more complex billing system, making it far more useful.

```
def calculate_power_bill(units_consumed):  
    """  
    Calculates a tiered power bill based on units consumed, a fixed charge, and GST  
    """  
    total_cost = 0  
  
    # Tiered pricing calculation  
    if units_consumed <= 100:  
        total_cost = units_consumed * 0.20  
    elif units_consumed <= 300:  
        total_cost = (100 * 0.20) + ((units_consumed - 100) * 0.25)  
    else:  
        total_cost = (100 * 0.20) + (200 * 0.25) + ((units_consumed - 300) * 0.30)  
  
    # Add fixed monthly charge  
    fixed_charge = 15  
    subtotal = total_cost + fixed_charge  
  
    # Add GST (18%)  
    gst_rate = 0.18  
    gst_amount = subtotal * gst_rate  
    final_bill = subtotal + gst_amount  
  
    return final_bill  
  
# Main part of the program  
try:  
    units = float(input("Enter the number of units consumed: "))  
    if units < 0:  
        print("Units consumed cannot be negative.")  
    else:  
        bill_amount = calculate_power_bill(units)  
        print(f"The total power bill for {units} units is: ${bill_amount:.2f}")  
  
except ValueError:  
    print("Invalid input. Please enter a numerical value.")
```

Task Description#4

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

Expected Output#4

- Consistent functions with shared logic

```
# 1. Import required libraries
# - hashlib for password hashing
# - json for file-based user store (optional)
# - os for checking file existence

# 2. Define a shared helper function: hash_password(password)
# - Input: plain password
# - Output: hashed password string
# - Purpose: keep registration and login consistent

# 3. Define a shared helper function: load_users(file_path)
# - Load users dict from JSON file
# - Return empty dict if file does not exist

# 4. Define a shared helper function: save_users(users, file_path)
# - Save users dict to JSON file

# 5. Function: register_user(username, password, file_path="users.json")
# - Validate username/password are non-empty strings
# - Load current users
# - Check if username already exists -> return failure
# - Hash password using hash_password
# - Add username:hashed_password to users dict
# - Save users
# - Return success message

# 6. Function: login_user(username, password, file_path="users.json")
# - Validate username/password are non-empty strings
# - Load current users
# - Check if username exists -> return failure if not
# - Hash input password and compare with stored hash
# - Return success/failure message
```



```

import hashlib
import json
import os

# -----
# Shared helper functions
# -----

def hash_password(password: str) -> str:
    """Hash a password using SHA-256"""
    return hashlib.sha256(password.encode()).hexdigest()

def load_users(file_path: str = "users.json") -> dict:
    """Load users from a JSON file"""
    if not os.path.exists(file_path):
        return {}
    with open(file_path, "r") as f:
        return json.load(f)

def save_users(users: dict, file_path: str = "users.json") -> None:
    """Save users to a JSON file"""
    with open(file_path, "w") as f:
        json.dump(users, f, indent=2)

```

```

def register_user(username: str, password: str, file_path: str = "users.json") -> dict:
    """Register a new user"""
    if not username or not password:
        return {"success": False, "message": "Username and password cannot be empty."}

    users = load_users(file_path)
    if username in users:
        return {"success": False, "message": "Username already exists."}

    users[username] = hash_password(password)
    save_users(users, file_path)
    return {"success": True, "message": f"User '{username}' registered successfully."}

def login_user(username: str, password: str, file_path: str = "users.json") -> dict:
    """Login an existing user"""
    if not username or not password:
        return {"success": False, "message": "Username and password cannot be empty."}

    users = load_users(file_path)
    if username not in users:
        return {"success": False, "message": "Username does not exist."}

    if users[username] != hash_password(password):
        return {"success": False, "message": "Incorrect password."}

    return {"success": True, "message": f"User '{username}' logged in successfully."}

```

```
# -----
# Example usage
# -----

if __name__ == "__main__":
    print(register_user("alice", "password123"))
    print(login_user("alice", "password123"))
    print(login_user("bob", "password123"))
```

Task Description#5

- Analyzing Prompt Specificity: Improving Temperature Conversion Function with

Clear Instructions

Expected Output#5

- Code quality difference analysis for various prompts

Step 0 — Vague Prompt (low specificity)

Prompt v0:

"Write Python code to convert temperature."

AI Output v0:

```
python

temp = float(input("Enter temperature: "))
unit = input("Enter unit (C/F): ")
if unit == "C":
    converted = (temp * 9/5) + 32
    print("Temperature in Fahrenheit:", converted)
elif unit == "F":
    converted = (temp - 32) * 5/9
    print("Temperature in Celsius:", converted)
else:
    print("Unknown unit")
```

Analysis:

- Works for basic input.
- Lacks **function structure**, **input validation**, **docstring**, or **unit testing**.
- Not reusable in other code.

Step 1 — Slightly clearer prompt (medium specificity)

Prompt v1:

"Write a Python function `convert_temperature(value, unit)` that converts between Celsius and Fahrenheit.
The function should return the converted temperature. Include basic input validation."

AI Output v1:

```
python Copy  
  
def convert_temperature(value, unit):  
    """  
    Convert temperature between Celsius and Fahrenheit.  
  
    Parameters:  
    - value: float, temperature to convert  
    - unit: str, "C" for Celsius, "F" for Fahrenheit  
  
    Returns:  
    - float: converted temperature  
  
    Raises:  
    - ValueError if unit is invalid  
    """  
    if unit == "C":  
        return (value * 9/5) + 32  
    elif unit == "F":  
        return (value - 32) * 5/9  
    else:  
        raise ValueError("Invalid unit. Use 'C' or 'F'.")
```

```
# Example usage  
print(convert_temperature(0, "C")) # 32.0  
print(convert_temperature(32, "F")) # 0.0
```

Copy Edit

Analysis:

- Better **code quality**: function encapsulation, docstring, exception handling.
- Still minimal: no rounding, no flexibility for case-insensitive units.

Step 2 — Highly specific prompt (high specificity)

Prompt v2:

"Write a Python function `convert_temperature(value: float, unit: str) -> float` that converts temperature between Celsius and Fahrenheit.

Requirements:

1. "Accept `"C"` or `"F"` (case-insensitive)."
2. "Return float rounded to 2 decimal places."
3. "Include proper docstring and type hints."
4. "Raise `ValueError` for invalid input."
5. "Include example usage and minimal tests."


```
def convert_temperature(value: float, unit: str) -> float:
    """
    Convert temperature between Celsius and Fahrenheit.

    Parameters
    -----
    value : float
        Temperature to convert
    unit : str
        "C" for Celsius, "F" for Fahrenheit (case-insensitive)

    Returns
    -----
    float
        Converted temperature, rounded to 2 decimals

    Raises
    -----
    ValueError
        If unit is not "C" or "F"
    """
    unit = unit.upper()
    if unit == "C":
        converted = (value * 9/5) + 32
    elif unit == "F":
        converted = (value - 32) * 5/9
    else:
        raise ValueError("Invalid unit. Use 'C' or 'F'.")
    return round(converted, 2)
```

Example usage

```
if __name__ == "__main__":
    print(convert_temperature(0, "c"))    # 32.0
    print(convert_temperature(32, "F"))   # 0.0
    # Minimal tests
    assert convert_temperature(100, "C") == 212.0
    assert convert_temperature(212, "F") == 100.0
```

Step 3 — Comparison Table

Prompt Specificity	Code Quality	Features
Vague (v0)	Low	Basic script, inline I/O, no function, no validation
Medium (v1)	Medium	Function, docstring, exception handling, reusable
High (v2)	High	Function, docstring, type hints, case-insensitive, rounding, example usage, tests

