

Task 1:

Try 3 different prompts to generate a factorial function.

prompt1:Write a python program to find the factorial of the given number.

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

Args:
n: The non-negative integer.

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

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

EXPLANATION:

This code defines a function called factorial that calculates the factorial of a non-negative integer using recursion.

Here's a breakdown:

def factorial(n):: This line defines the function named factorial that takes one argument n.

Docstring: The triple-quoted string explains what the function does, its arguments (Args), and what it returns (Returns).

if n == 0:: This is the base case of the recursion. The factorial of 0 is 1, so if n is 0, the function returns 1.

else:: This is the recursive step. If n is not 0, the function returns n multiplied by the factorial of n-1. This calls the function itself with a smaller value of n, moving closer to the base case.

num = 5: This line assigns the value 5 to the variable num.

result = factorial(num): This line calls the factorial function with num (which is 5) and stores the returned value (the factorial of 5) in the result variable.

print(f"The factorial of {num} is {result}"): This line prints the result in a formatted string

PROMPT2:Python code for finding factorial of number

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

 Args:
  n: The non-negative integer.

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

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

PROMPT3:Factorial python code

```
def factorial(n):
 """Calculates the factorial of a non-negative integer."""
 if n == 0:
  return 1
 else:
  return n * factorial(n-1)

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

Task2:

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

PROMPT:

Write a Python function that sorts a list of numbers in ascending order.

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

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

```
def sort_list_ascending(input_list):
 """Sorts a list of numbers in ascending order.

 Args:
  input_list: A list of numbers.
```

```
    Returns:
      A new list with the numbers sorted in ascending order.
    """
    return sorted(input_list)

# Example usage
input_list = [3, 1, 4, 1, 5, 9, 2, 6]
output_list = sort_list_ascending(input_list)
print(f"Input: {input_list}")
print(f"Output: {output_list}")
```

This code defines a Python function sort_list_ascending that takes a list of numbers as input and returns a new list with the numbers sorted in ascending order.

Here's a breakdown:

def sort_list_ascending(input_list):: This line defines the function named sort_list_ascending that accepts one argument, input_list.

Docstring: The triple-quoted string explains the purpose of the function, its arguments (Args), and what it returns (Returns).

return sorted(input_list): This is the core of the function. It uses the built-in Python function sorted() to create a new sorted list from the input_list and returns this new list. The sorted() function by default sorts in ascending order.

Example usage: This is a comment indicating the start of the example usage of the function.

input_list = [3, 1, 4, 1, 5, 9, 2, 6]: This line creates a list of numbers and assigns it to the variable input_list.

output_list = sort_list_ascending(input_list): This line calls the sort_list_ascending function with the input_list and stores the returned sorted list in the output_list variable.

print(f"Input: {input_list}"): This line prints the original input list.

print(f"Output: {output_list}"): This line prints the sorted output list.

Task 3:Start with the vague prompt “Generate python code to calculate power bill” and improve it step-by-step

```
import random

def calculate_power_bill_random():
 """Calculates a power bill using random consumption and a basic rate."""
 # Generate a random electricity consumption value (e.g., between 50 and 500 kWh)
 consumption = random.uniform(50, 500)
 # Define a basic rate per kWh (e.g., $0.15)
 rate_per_kwh = 0.15

 # Calculate the bill
 bill_amount = consumption * rate_per_kwh

 return consumption, bill_amount

# Calculate the bill with random values
consumption, bill = calculate_power_bill_random()

print(f"Electricity Consumption: {consumption:.2f} kWh")
print(f"Basic Rate per kWh: $0.15")
print(f"Estimated Power Bill: ${bill:.2f}")
```

EXPLANATION:

This code calculates a basic power bill using a randomly generated electricity consumption and a fixed rate.

Here's a breakdown:

import random: This line imports the random module, which is needed to generate random numbers.

def calculate_power_bill_random():: This defines a function named calculate_power_bill_random.

Docstring: Explains what the function does.

consumption = random.uniform(50, 500): This line generates a random floating-point number between 50 and 500 (inclusive) and assigns it to the consumption variable. This simulates a random electricity consumption in kWh.

rate_per_kwh = 0.15: This line defines a fixed basic rate of $0.15 per kWh.

bill_amount = consumption * rate_per_kwh: This calculates the bill amount by multiplying the consumption by the rate.

return consumption, bill_amount: The function returns both the generated consumption and the calculated bill amount.

consumption, bill = calculate_power_bill_random(): This line calls the function and unpacks the returned consumption and bill amount into respective variables.

print(...): These lines print the calculated electricity consumption, the basic rate, and the estimated power bill, formatted to two decimal places.

TASK4:Write structured comments to help AI generate two linked functions (e.g., login_user()

and register_user()).

Simple dictionary to simulate user data storage

In a real application, this would be a database or other secure storage

user_data = {}

def register_user(username, password):

"""Registers a new user.

```
    Args:
      username: The desired username.
      password: The desired password.

    Returns:
      True if registration is successful, False otherwise.
    """
    if username in user_data:
      print(f"Username '{username}' already exists.")
      return False
    else:
      # In a real application, the password should be hashed and stored securely
      user_data[username] = password
      print(f"User '{username}' registered successfully.")
      return True

def login_user(username, password):
    """Logs in an existing user.

    Args:
      username: The username.
      password: The password.

    Returns:
      True if login is successful, False otherwise.
    """
    if username in user_data and user_data[username] == password:
      print(f"User '{username}' logged in successfully.")
```

```
    return True
  else:
    print("Invalid username or password.")
    return False

# Example Usage:
register_user("testuser", "password123")
login_user("testuser", "password123")
login_user("anotheruser", "fakepassword")
```

This code provides a simple implementation of user registration and login functionality using a Python dictionary to store user data.

Here's a breakdown:

user_data = {}: This initializes an empty dictionary named user_data. This dictionary is used to store usernames as keys and their corresponding passwords as values. The comments emphasize that this is a simplified approach for demonstration and a real application would use a more secure storage method like a database with hashed passwords.

def register_user(username, password):: This defines the register_user function which takes username and password as input.

The docstring explains the purpose, arguments, and return value of the function.

if username in user_data:: This checks if the provided username already exists as a key in the user_data dictionary.

If the username exists, it prints a message and returns False to indicate that registration failed.

else:: If the username does not exist, it proceeds with registration.

user_data[username] = password: This line adds the new username and password as a key-value pair to the user_data dictionary. The comment again highlights the need for secure password handling in a real application.

It prints a success message and returns True.

def login_user(username, password):: This defines the login_user function which also takes username and password as input.

The docstring explains the purpose, arguments, and return value.

if username in user_data and user_data[username] == password:: This checks two conditions: if the username exists in user_data AND if the provided password matches the password stored for that username in the dictionary.

If both conditions are true, it prints a success message and returns True to indicate successful login.

else:: If either condition is false, it prints an "Invalid username or password" message and returns False.

Example Usage:: This section shows how to use the defined functions with example calls to register_user and login_user to demonstrate their functionality.

TASK5:Analyzing Prompt Specificity: Improving Temperature Conversion Function with

Clear Instructions

PROMPT:Write a python function of temperature conversion

```
def celsius_to_fahrenheit(celsius):
  """Converts Celsius to Fahrenheit."""
  return (celsius * 9/5) + 32


def fahrenheit_to_celsius(fahrenheit):
  """Converts Fahrenheit to Celsius."""
  return (fahrenheit - 32) * 5/9


def celsius_to_kelvin(celsius):
  """Converts Celsius to Kelvin."""
  return celsius + 273.15


def kelvin_to_celsius(kelvin):
  """Converts Kelvin to Celsius."""
  return kelvin - 273.15
```

```
def fahrenheit_to_kelvin(fahrenheit):
  """Converts Fahrenheit to Kelvin."""
  celsius = fahrenheit_to_celsius(fahrenheit)
  return celsius_to_kelvin(celsius)

def kelvin_to_fahrenheit(kelvin):
  """Converts Kelvin to Fahrenheit."""
  celsius = kelvin_to_celsius(kelvin)
  return celsius_to_fahrenheit(celsius)

# Example usage:
celsius_temp = 25
fahrenheit_temp = celsius_to_fahrenheit(celsius_temp)
print(f"{celsius_temp}°C is {fahrenheit_temp:.2f}°F")

fahrenheit_temp = 77
celsius_temp = fahrenheit_to_celsius(fahrenheit_temp)
print(f"{fahrenheit_temp}°F is {celsius_temp:.2f}°C")

celsius_temp = 0
kelvin_temp = celsius_to_kelvin(celsius_temp)
print(f"{celsius_temp}°C is {kelvin_temp:.2f}K")
```

This code defines several Python functions for converting temperatures between Celsius, Fahrenheit, and Kelvin.

Here's a breakdown of each function:

celsius_to_fahrenheit(celsius): Converts a temperature from Celsius to Fahrenheit using the formula (Celsius * 9/5) + 32.

fahrenheit_to_celsius(fahrenheit): Converts a temperature from Fahrenheit to Celsius using the formula (Fahrenheit - 32) * 5/9.

celsius_to_kelvin(celsius): Converts a temperature from Celsius to Kelvin by adding 273.15.

kelvin_to_celsius(kelvin): Converts a temperature from Kelvin to Celsius by subtracting 273.15.

fahrenheit_to_kelvin(fahrenheit): Converts Fahrenheit to Kelvin by first converting Fahrenheit to Celsius using fahrenheit_to_celsius() and then converting the resulting Celsius temperature to Kelvin using celsius_to_kelvin().

kelvin_to_fahrenheit(kelvin): Converts Kelvin to Fahrenheit by first converting Kelvin to Celsius using kelvin_to_celsius() and then converting the resulting Celsius temperature to Fahrenheit using celsius_to_fahrenheit().