AI ASSISTED CODING

Roll no.: 2503A51L41

Batch: 24BTCAICSB20

Name: G Hasini

Lab 3: Prompt Engineering – Improving Prompts and Context Management

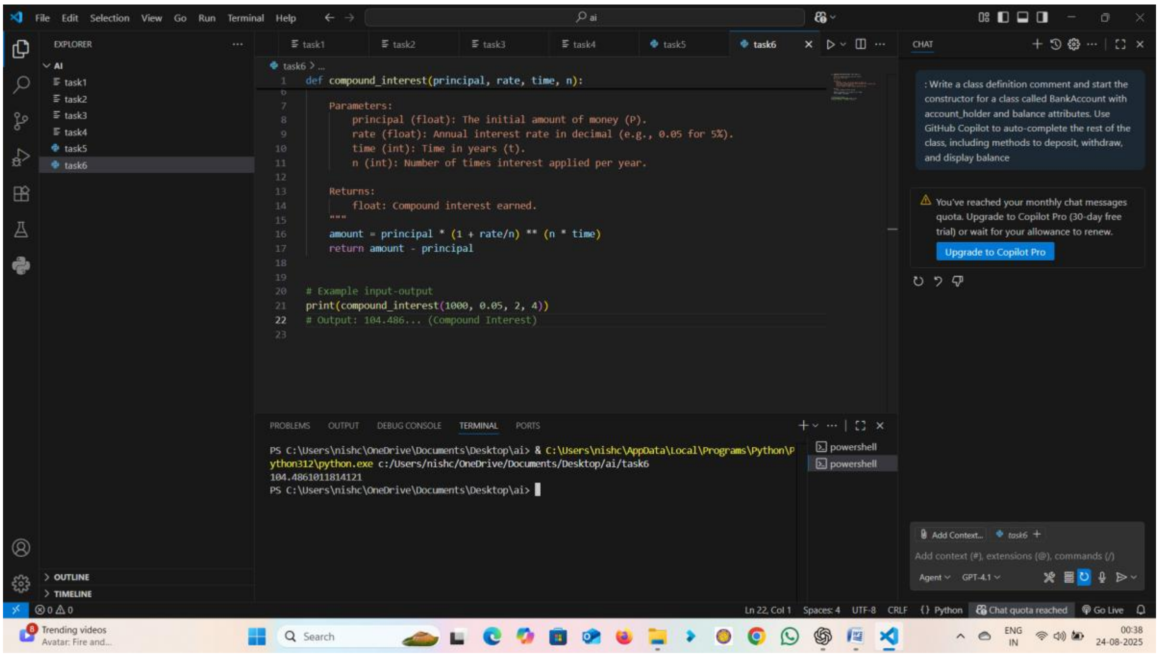
Task#1

Prompt: Ask AI to write a function to calculate compound interest, starting with only the function name. Then add a docstring, then input-output example.

Code generated:

AI Generated Code:

```
# Step 1
def compound_interest():
    pass
# Step 2 - with docstring
def compound_interest(principal, rate, time, n):
    """
    Calculate compound interest using formula:
    A = P * (1 + r/n)^(n*t)
    CI = A - P
    """
# Step 3 - with full logic and example
def compound_interest(principal, rate, time, n):
    amount = principal * (1 + rate/n)**(n*time)
    return amount - principal
print(compound_interest(1000, 0.05, 2, 4))  # 104.486..
```



```
def compound_interest(principal, rate, time, n):

    Parameters:
        principal (float): The initial amount of money (P).
        rate (float): Annual interest rate in decimal (e.g., 0.05 for 5%).
        time (int): Time in years (t).
        n (int): Number of times interest applied per year.

    Returns:
        float: Compound interest earned.
    """
    amount = principal * (1 + rate/n) ** (n * time)
    return amount - principal


# Example input-output
print(compound_interest(1000, 0.05, 2, 4))
# Output: 104.486... (Compound Interest)
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\P
ython312\python.exe c:/Users/nishc/OneDrive/Documents/Desktop/ai/task6
104.4861011814121
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

⚠ You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Output: Input: P=1000, r=5%, t=2 years, n=4 → **Output**: 104.486 (compound interest)

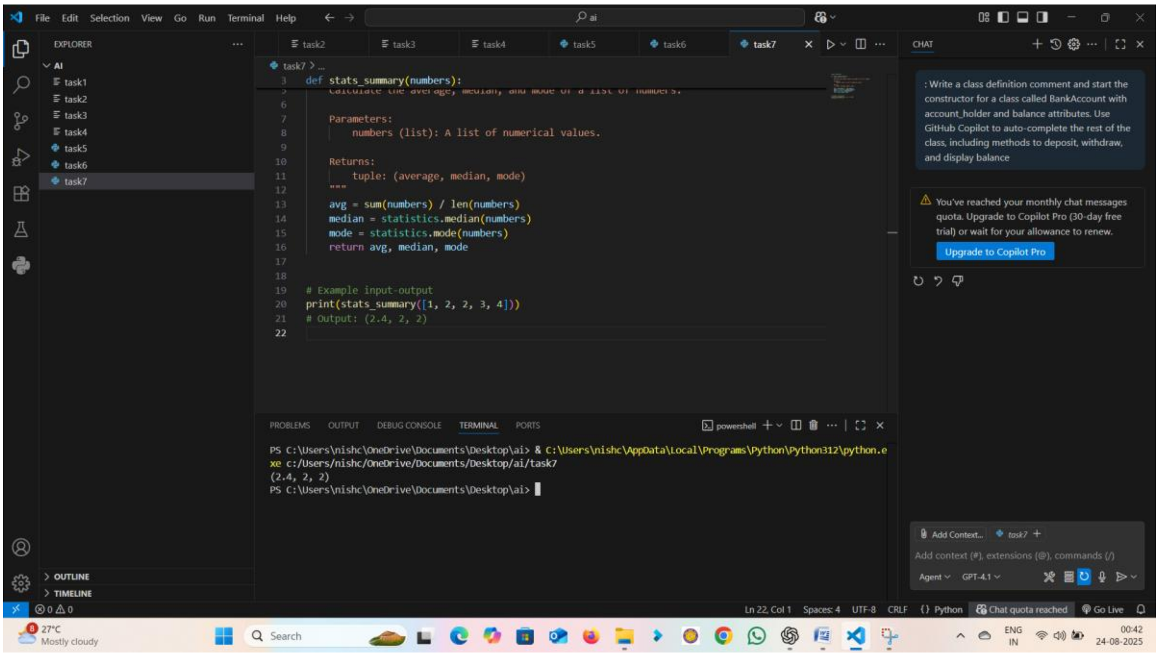
Observation: The task showed how step-by-step prompting helps in building a complete function. Starting with only the function name, then adding a docstring, and finally implementing the logic, made the AI produce accurate and clear code for calculating compound interest.

Task#2

Prompt: Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.

AI Generated Code:

```
import statistics
def stats_summary(numbers):
   avg = sum(numbers)/len(numbers)
   median = statistics.median(numbers)
   mode = statistics.mode(numbers)
   return avg, median, mode
print(stats_summary([1,2,2,3,4]))  # (2.4, 2, 2)
```



```
def stats_summary(numbers):
    Calculate the average, median, and mode of a list of numbers.

    Parameters:
        numbers (list): A list of numerical values.

    Returns:
        tuple: (average, median, mode)
    """
    avg = sum(numbers) / len(numbers)
    median = statistics.median(numbers)
    mode = statistics.mode(numbers)
    return avg, median, mode


# Example input-output
print(stats_summary([1, 2, 2, 3, 4]))
# Output: (2.4, 2, 2)
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.e
xe c:/Users/nishc/OneDrive/Documents/Desktop/ai/task7
(2.4, 2, 2)
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

⚠ You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Output: Input: [1,2,2,3,4] → **Output**: Average=2.4, Median=2, Mode=2

Observation: A vague prompt produced poor results, but refining it led to a precise function that correctly calculates average, median, and mode. This highlights the importance of giving clear, specific prompts to obtain reliable code.

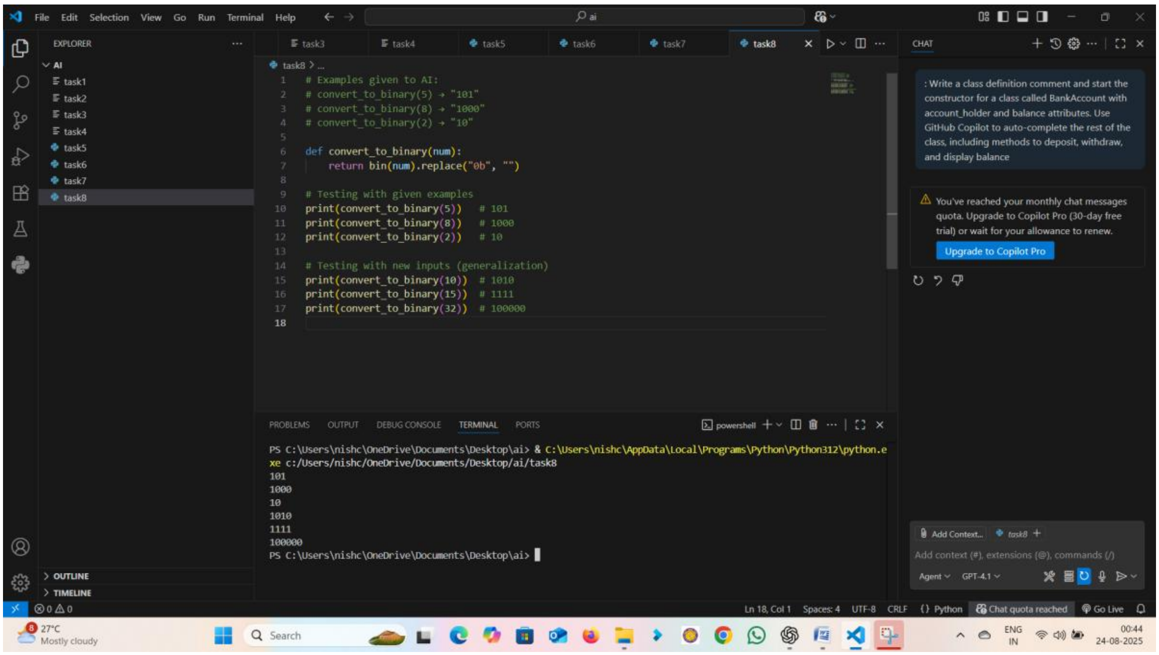
Task#3

Prompt: Provide multiple examples of input-output to the AI for convert_to_binary(num) function. Observe how AI uses few-shot prompting to generalize.

AI Generated Code:

```
def convert_to_binary(num):
   return bin(num).replace("0b", "")
print(convert_to_binary(5))  # 101

print(convert_to_binary(8))  # 1000
print(convert_to_binary(2))  # 10
```



```python
# Examples given to AI:
# convert_to_binary(5) → "101"
# convert_to_binary(8) → "1000"
# convert_to_binary(2) → "10"

def convert_to_binary(num):
    return bin(num).replace("0b", "")

# Testing with given examples
print(convert_to_binary(5))   # 101
print(convert_to_binary(8))   # 1000
print(convert_to_binary(2))   # 10

# Testing with new inputs (generalization)
print(convert_to_binary(10))  # 1010
print(convert_to_binary(15))  # 1111
print(convert_to_binary(32))  # 100000
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.e
xe c:/Users/nishc/OneDrive/Documents/Desktop/ai/task8
101
1000
10
1010
1111
100000
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

⚠ You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Output: Input: 5 → Output: 101

Input: 8 → Output: 1000

Input: 2 → Output: 10

Input:10->output:1010

Input:15->output:1111

Input:32->output:100000

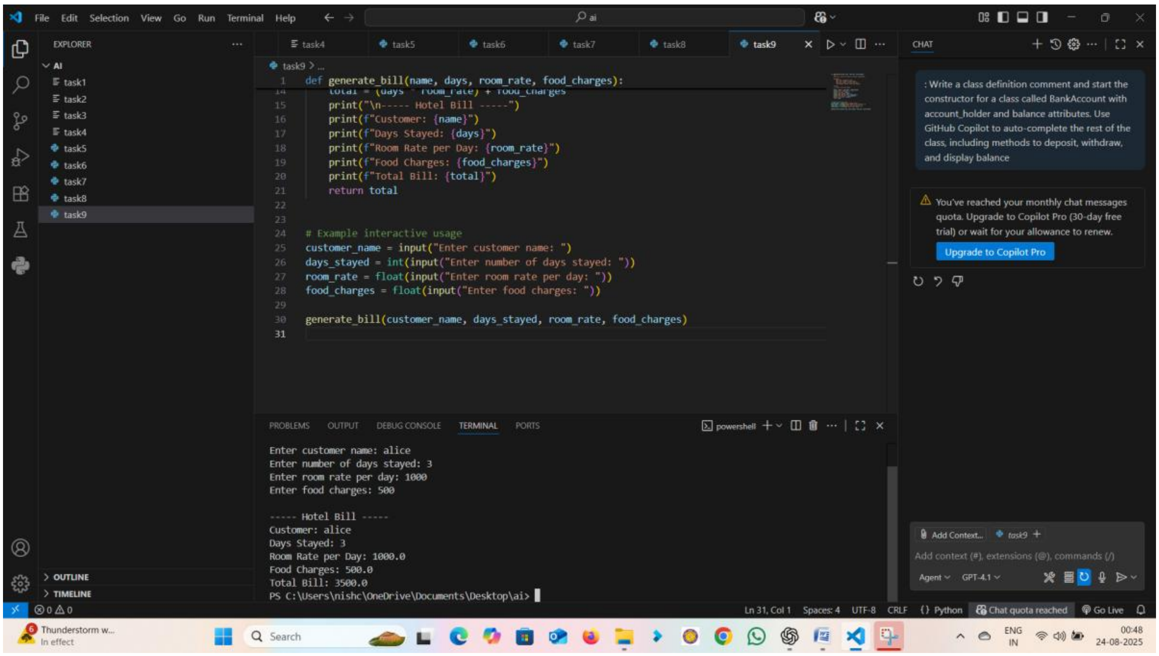
Observation: By providing examples, the AI learned the expected output format and generated correct code using Python's built-in bin() function. Few-shot prompting improved accuracy and reduced ambiguity.

Task#4

Prompt: Create an user interface for an hotel to generate bill based on customer requirements.

AI generated code:

```
def generate_bill(name, days, room_rate, food_charges):
    total = (days * room_rate) + food_charges
    print("----- Hotel Bill -----")
    print(f"Customer: {name}")
    print(f"Days Stayed: {days}")
    print(f"Room Rate: {room_rate}")
    print(f"Food Charges: {food_charges}")
    print(f"Total Bill: {total}")
    return total
generate_bill("Alice", 3, 1000, 500)
```



```
def generate_bill(name, days, room_rate, food_charges):
    total = (days * room_rate) + food_charges
    print("\n----- Hotel Bill -----")
    print(f"Customer: {name}")
    print(f"Days Stayed: {days}")
    print(f"Room Rate per Day: {room_rate}")
    print(f"Food Charges: {food_charges}")
    print(f"Total Bill: {total}")
    return total


# Example interactive usage
customer_name = input("Enter customer name: ")
days_stayed = int(input("Enter number of days stayed: "))
room_rate = float(input("Enter room rate per day: "))
food_charges = float(input("Enter food charges: "))

generate_bill(customer_name, days_stayed, room_rate, food_charges)
```

```
Enter customer name: alice
Enter number of days stayed: 3
Enter room rate per day: 1000
Enter food charges: 500

----- Hotel Bill -----
Customer: alice
Days Stayed: 3
Room Rate per Day: 1000.0
Food Charges: 500.0
Total Bill: 3500.0
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

⚠ You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Output: Customer: Alice, Days=3, Room Rate=1000/day, Food=500 → Total Bill = 3500

Observation: The AI generated a simple, text-based billing system that calculates total charges and prints a formatted bill. This shows how descriptive prompts can produce practical, user-friendly solutions tailored to real-world applications.

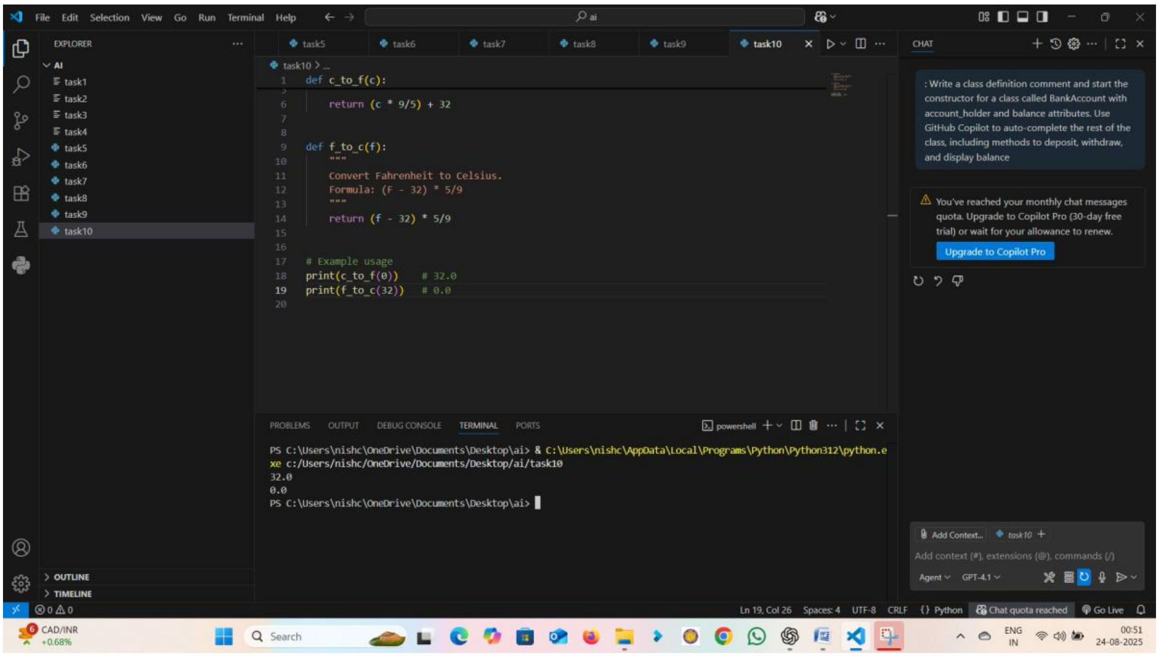
Task #5

Prompt: Analyzing Prompt Specificity: Improving Temperature Conversion Function with Clear Instructions.

AI Generated Code:

```
def c_to_f(c):
    return (c * 9/5) + 32
def f_to_c(f):
    return (f - 32) * 5/9
```

```
print(c_to_f(0))   # 32.0
print(f_to_c(32))  # 0.0
```



```python
def c_to_f(c):

    return (c * 9/5) + 32


def f_to_c(f):
    """
    Convert Fahrenheit to Celsius.
    Formula: (F - 32) * 5/9
    """
    return (f - 32) * 5/9


# Example usage
print(c_to_f(0))    # 32.0
print(f_to_c(32))   # 0.0
```

```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\Python312\python.e
xe c:/Users/nishc/OneDrive/Documents/Desktop/ai/task10
32.0
0.0
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

⚠ You've reached your monthly chat messages quota. Upgrade to Copilot Pro (30-day free trial) or wait for your allowance to renew.

Output: Input: 0°C → Output: 32°F

Input: 32°F → Output: 0°C

Observation: The vague prompt gave incomplete or unclear results, while the specific prompt resulted in accurate conversion functions for Celsius and Fahrenheit. This demonstrates that prompt specificity directly affects the quality of AI-generated output.