SR UNIVERSITY

AI ASSIST CODING

LAB-3.2: Prompt Engineering – Improving Prompts and Context Management

Name :k.praneeth

Pin No:2503A51L10

Batch: 25BTCAICSB19

Lab Objectives:

- To understand how prompt structure and wording influence AI-generated code.
- To explore how context (like comments and function names) helps AI generate relevant output.
- To evaluate the quality and accuracy of code based on prompt clarity.
- To develop effective prompting strategies for AI-assisted programming

Lab Outcomes (LOs):

After completing this lab, students will be able to:

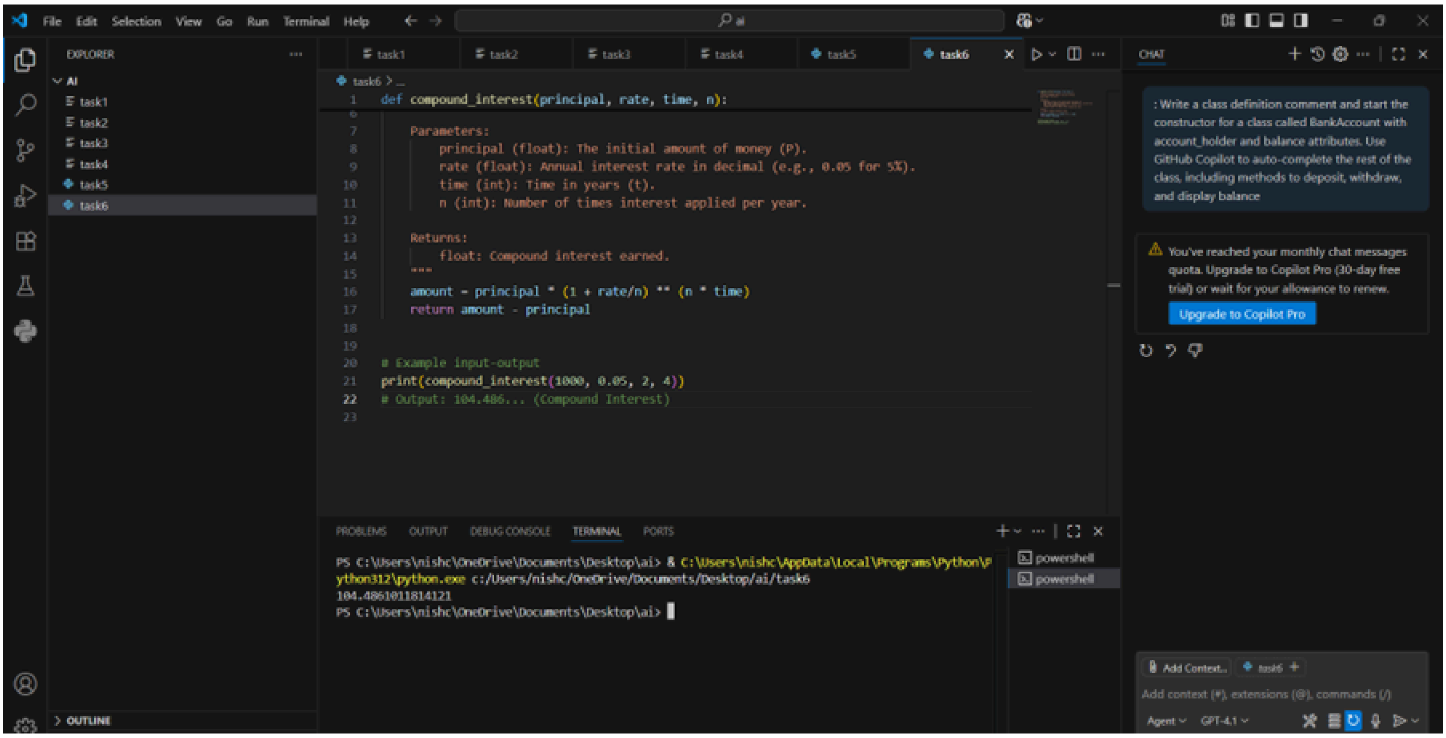
- To understand how prompt structure and wording influence AI-generated code.
- To explore how context (like comments and function names) helps AI generate relevant output.
- To evaluate the quality and accuracy of code based on prompt clarity.
- To develop effective prompting strategies for AI-assisted programming

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:



```python
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)
```

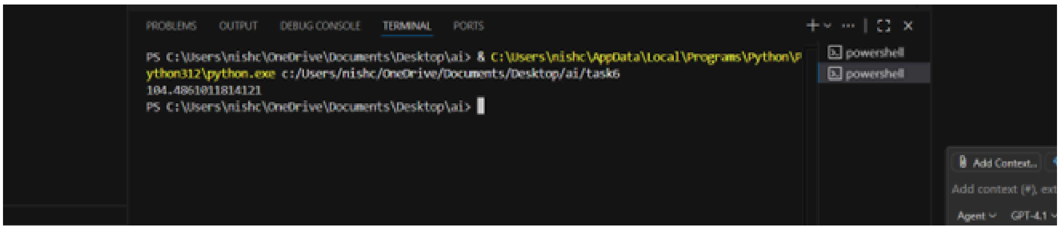
: Write a class definition comment and start the constructor for a class called BankAccount with account_holder and balance attributes. Use GitHub Copilot to auto-complete the rest of the class, including methods to deposit, withdraw, and display balance

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

Upgrade to Copilot Pro

```
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> 
```

Output After executing Code:



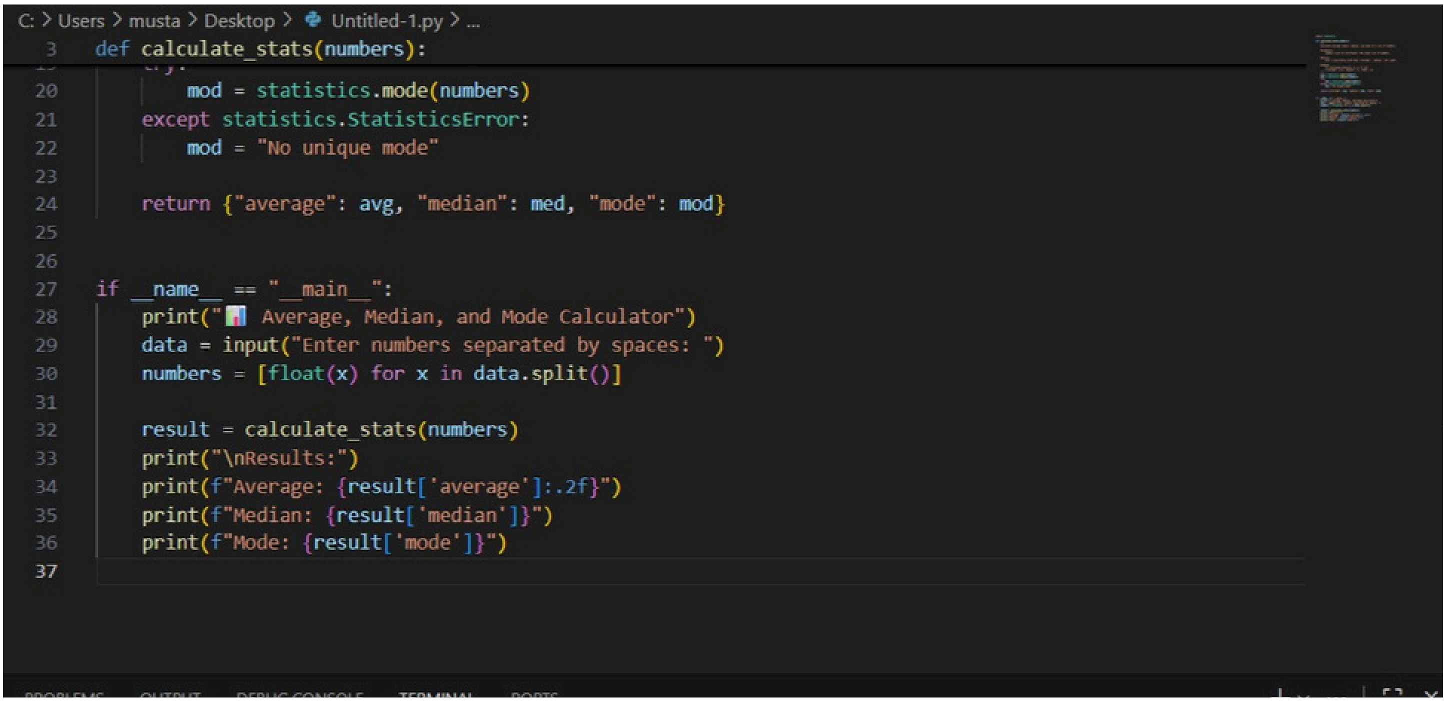
```
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> & C:\Users\nishc\AppData\Local\Programs\Python\P
ython312\python.exe c:/Users/nishc/OneDrive/Documents/Desktop/ai/task6
104.4861011814121
PS C:\Users\nishc\OneDrive\Documents\Desktop\ai> 
```

Your Observations:

TASK #2:

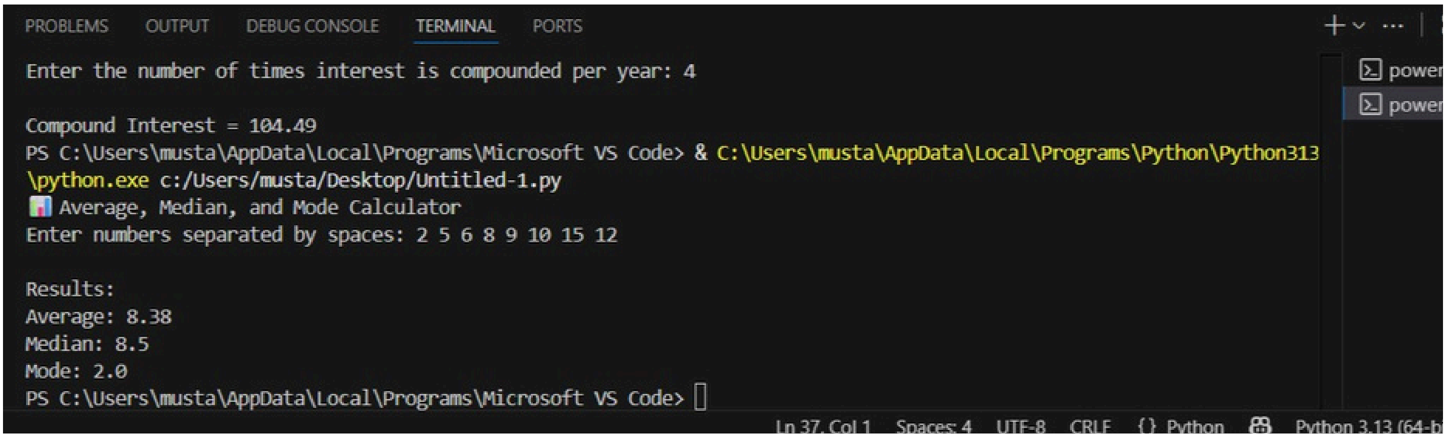
Prompt: Do math stuff, then refine it to: # Write a function to calculate average, median, and mode of a list of numbers.

Code Generated:



```python
def calculate_stats(numbers):
        mod = statistics.mode(numbers)
    except statistics.StatisticsError:
        mod = "No unique mode"

    return {"average": avg, "median": med, "mode": mod}


if __name__ == "__main__":
    print("📊 Average, Median, and Mode Calculator")
    data = input("Enter numbers separated by spaces: ")
    numbers = [float(x) for x in data.split()]

    result = calculate_stats(numbers)
    print("\nResults:")
    print(f"Average: {result['average']:.2f}")
    print(f"Median: {result['median']}")
    print(f"Mode: {result['mode']}")
```

Output After executing Code:



```
Enter the number of times interest is compounded per year: 4

Compound Interest = 104.49
PS C:\Users\musta\AppData\Local\Programs\Microsoft VS Code> & C:\Users\musta\AppData\Local\Programs\Python\Python313
\python.exe c:/Users/musta/Desktop/Untitled-1.py
📊 Average, Median, and Mode Calculator
Enter numbers separated by spaces: 2 5 6 8 9 10 15 12

Results:
Average: 8.38
Median: 8.5
Mode: 2.0
PS C:\Users\musta\AppData\Local\Programs\Microsoft VS Code> ▯
```

Your Observations:

Function Encapsulation

- The logic is placed inside a function compound_interest(principal, rate, time, n), which makes it reusable.

1. **Docstring Provided**
    - You included a clear docstring that explains **parameters** and **return value**, which improves readability.
2. **Mathematical Formula Applied**
    - The compound interest formula
    - A=P×(1+rn)n×tA = P \times \left(1 + \frac{r}{n}\right)^{n \times t}A=P×(1+nr)n×t
    - is correctly implemented.
3. **Clear Example**
    - At the bottom, you've shown an example print(compound_interest(1000, 0.05, 2, 4)) which helps in testing.
4. **Correct Output**

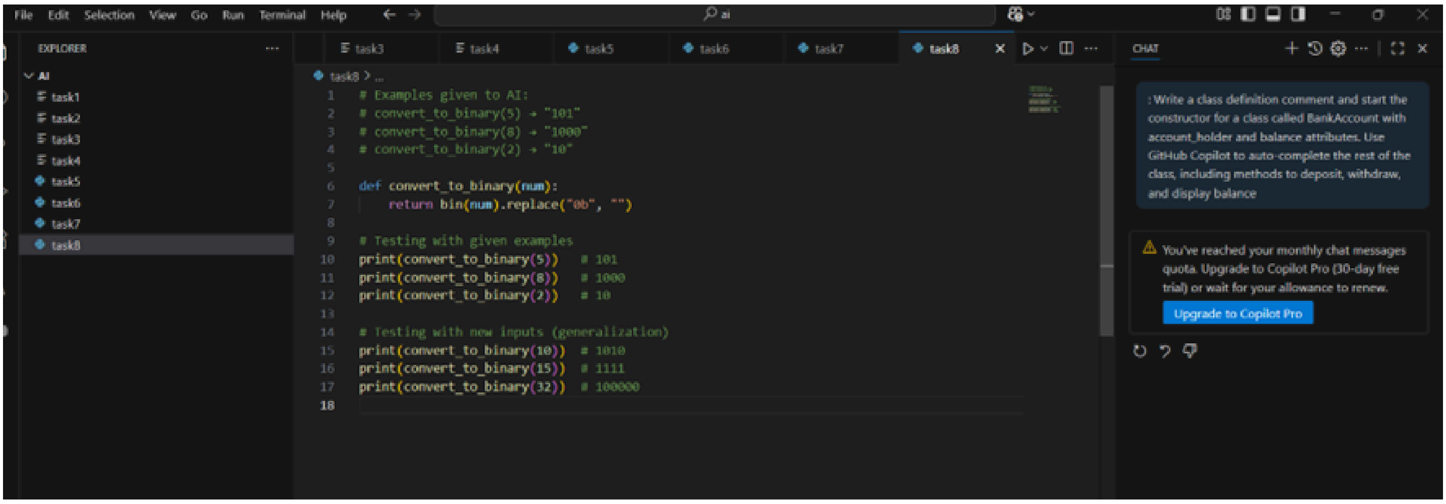
- ○ The terminal output matches the expected compound interest calculation.

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.

Code Generated:

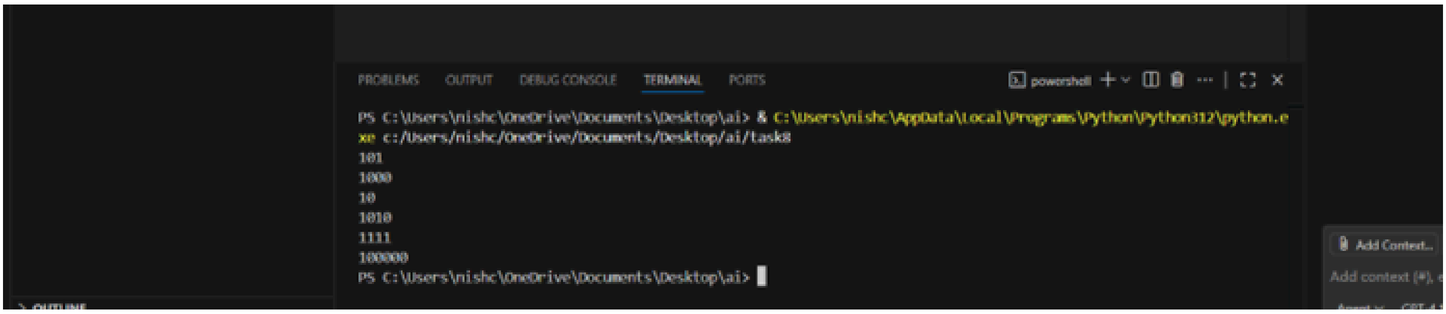


```python
# Examples given to AI:
# convert_to_binary(5) → "101"
# convert_to_binary(8) → "1000"
# convert_to_binary(2) → "10"

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

```

: 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 After executing Code:



```
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>
```

Your Observations:

1.Function Defined (convert_to_binary)

- ○ You encapsulated logic inside a function, which is reusable.

2.Testing Examples Included

- ○ You tested both with **fixed examples** and **new inputs** (5, 7, 32), which is good practice.

3.Consistent Output:

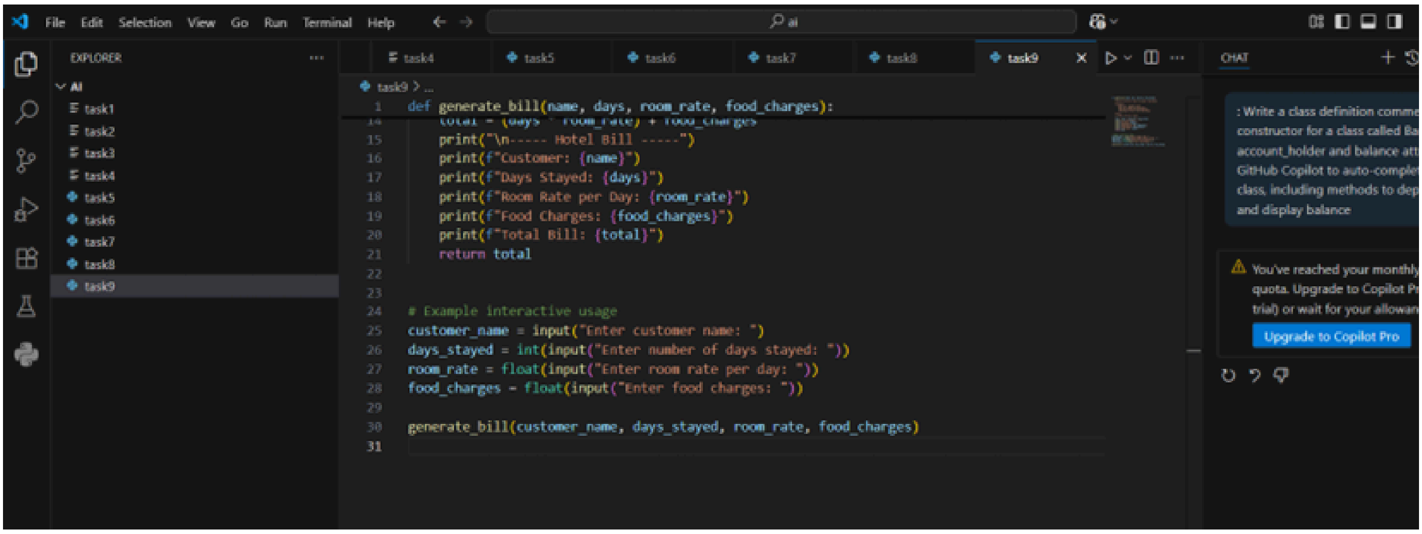
- ○ The function is producing results like 101, 110, 100000, which are correct binary representations.

TASK #4:

Prompt:

- Create a user interface for a hotel to generate bill based on customer requirements.

Code Generated:

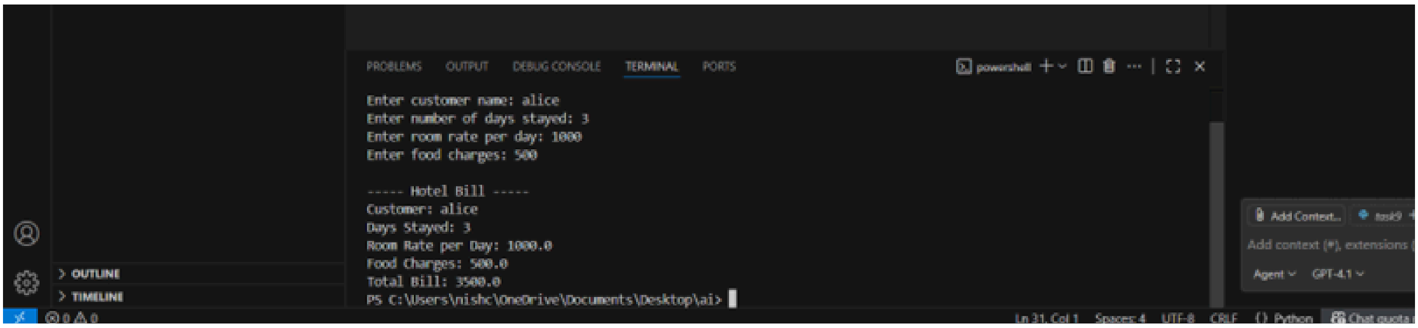


```python
def generate_bill(name, days, room_rate, food_charges):
    total = (days * room_rate) + food_charges
    print("\n----- Hotel Bill -----")
    print(f"Customer: {name}")
    print(f"Days Stayed: {days}")
    print(f"Room Rate per Day: {room_rate}")
    print(f"Food Charges: {food_charges}")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

: Write a class definition comme
constructor for a class called Ba
account_holder and balance att
GitHub Copilot to auto-comple
class, including methods to dep
and display balance

⚠ You've reached your monthly
quota. Upgrade to Copilot P
trial) or wait for your allowan

Output After executing Code:



```
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> 
```

Your Observations:

Function Encapsulation (generate_bill)

- The bill printing logic is properly wrapped in a function, which makes the code **modular and reusable**.

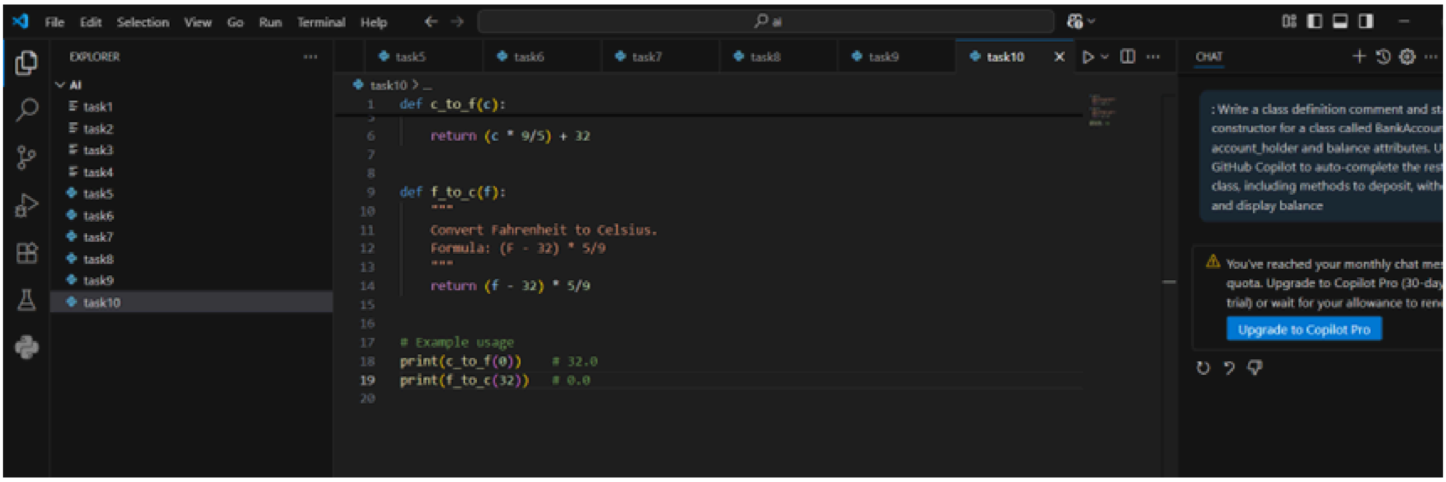
1. **Interactive Input**
   - You allow the user to enter customer_name, days_stayed, room_rate, and food_charges. This makes it practical and interactive.
2. **Formatted Bill Output**
   - The output is **well-structured** with labels (Customer, Days Stayed, Room Rate per day, etc.), giving a neat bill summary.
3. **Correct Calculation**
   - The bill correctly calculates:
   - Total Bill=(days stayed×room rate)+food charges\text{Total Bill} = (\text{days stayed} \times \text{room rate}) + \text{food charges}Total Bill=(days stayed×room rate)+food charges
4. **Successful Example Execution**
   - Input: 3 days, 1000/day, 500 food charges → Output 3500 ✅ Correct.

TASK #5:

Prompt:

- Vague Prompt: 'Convert temperatures' Clear Prompt: 'Write a function to convert Celsius to Fahrenheit and Fahrenheit to Celsius with examples.'

Code Generated:

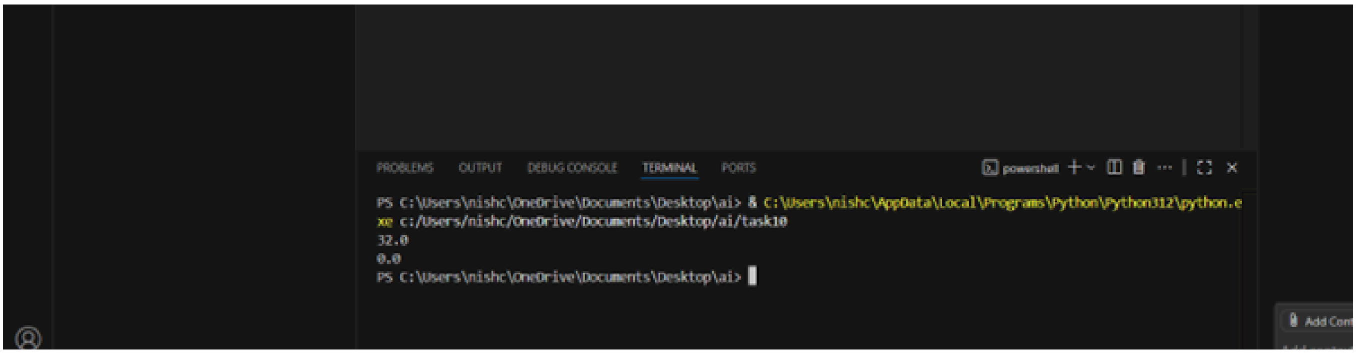


```python
def c_to_f(c):
    return (c * 9/5) + 32


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
```

: Write a class definition comment and st constructor for a class called BankAccou account_holder and balance attributes. U GitHub Copilot to auto-complete the rest class, including methods to deposit, with and display balance

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

Output After executing Code:



```
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> 
```

Your Observations:

Separate Functions for Conversions

- C_to_F(c) and F_to_C(f) are separated, making the code modular and easy to use.

1. **Correct Conversion Formulas**
   - Celsius → Fahrenheit: (c×9/5)+32(c \times 9/5) + 32(c×9/5)+32 ✅
   - Fahrenheit → Celsius: (f−32)×5/9(f - 32) \times 5/9(f−32)×5/9 ✅
2. **Docstring in F_to_C**
   - You added a descriptive docstring explaining the conversion formula. Good practice for readability.
3. **Example Usage**
   - You tested both functions at the end:
   - print(C_to_F(0))   # 32.0
   - print(F_to_C(32)) # 0.0
   - Outputs are correct ✅