

Lab 3: Prompt Engineering

Improving Prompts and Context Management

Course Code: 24CSBTB41

Course Title: AI Assistant Coding

Assignment No: 3.2

Roll Number: 2503A51L36

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 (LO's):

After completing this lab, students will be able to:

- Generate Python code using Google Gemini in Google Colab.
- Analyze the effectiveness of code explanations and suggestions by Gemini.
- Set up and use Cursor AI for AI-powered coding assistance.
- Evaluate and refactor code using Cursor AI features.
- Compare AI tool behavior and code quality across different platforms.

Task 1: Comparison of AI-generated code styles

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

Code:

```
def calculate_compound_interest(principal, rate, time, n=1):
```

```
    """
```

```
    Calculate compound interest.
```

```
    Args:
```

```
        principal (float): Initial amount of money invested or borrowed.
```

```
        rate (float): Annual interest rate in decimal (e.g., 0.05 for 5%).
```

```
        time (float): Time in years.
```

```
        n (int): Number of times interest is compounded per year.
```

```
    Returns:
```

```
        float: Final amount after compounding.
```

```
    Example:
```

```
    >>> calculate_compound_interest(1000, 0.05, 2, 4)
```

```
    1104.486...
```

```
    """
```

```
    amount = principal * (1 + rate / n) ** (n * time)
```

```
    return amount
```

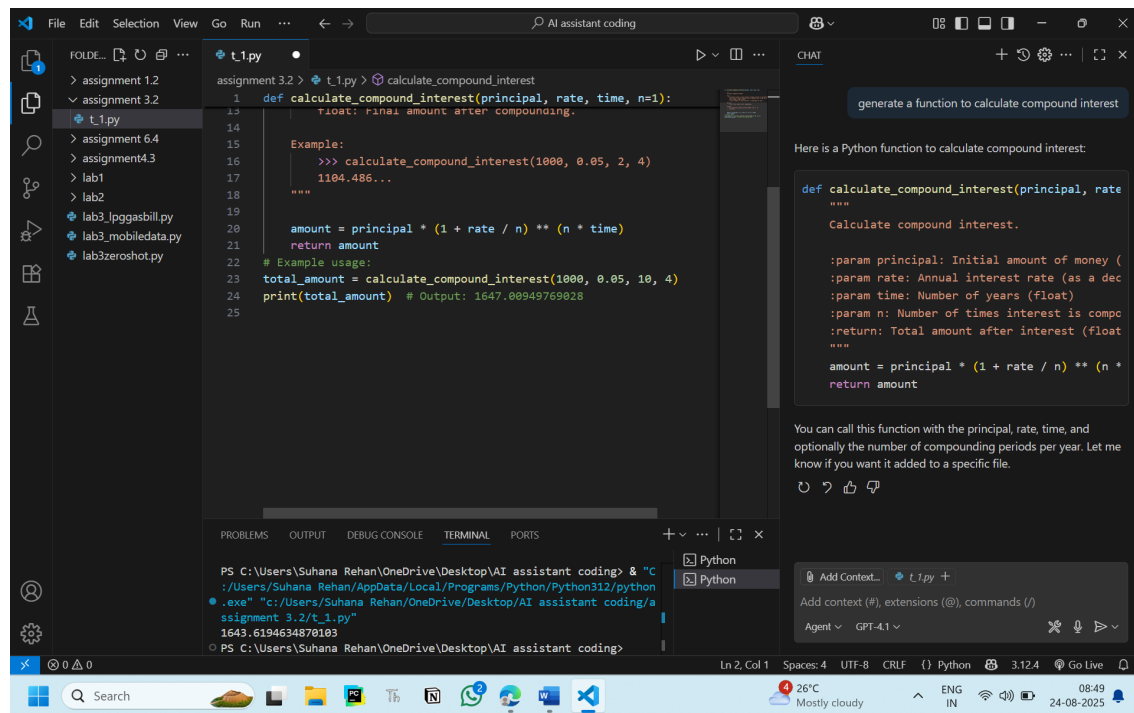
```
# Example usage:
```

```
total_amount = calculate_compound_interest(1000, 0.05, 10, 4)
```

```
print(total_amount) # Output: 1647.00949769028
```

OBSERVATION:

Here the both copilot and ai in the code generated the code, where copilot generates the according to our prompt and the ai in the code like to guess what are we going write next and generates the code



Expected Output:

1643.6194634870103

Task 2: Function evolves from unclear to accurate multi-statistical operation.

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

Code:

```
def calualte_sum(a, b):
    return a + b
def calculate_product(a, b):
    return a * b
# Example usage:
sum_result = calualte_sum(5, 10)
print(sum_result) # Output: 15
product_result = calculate_product(5, 10)
print(product_result) # Output: 50
# Write a function to calculate average, median, and mode of a list of numbers.
```

```

def calculate_average(numbers):
    return sum(numbers) / len(numbers)
def calculate_median(numbers):
    sorted_numbers = sorted(numbers)
    n = len(sorted_numbers)
    mid = n // 2
    if n % 2 == 0:
        return (sorted_numbers[mid - 1] + sorted_numbers[mid]) / 2
    else:
        return sorted_numbers[mid]
def calculate_mode(numbers):
    from collections import Counter
    count = Counter(numbers)
    max_count = max(count.values())
    modes = [k for k, v in count.items() if v == max_count]
    if len(modes) == len(count):
        return None # No mode
    return modes
# Example usage:
data = [1, 2, 2, 3, 4]
average = calculate_average(data)
median = calculate_median(data)
mode = calculate_mode(data)
print(f"Average: {average}, Median: {median}, Mode: {mode}") # Output: Average: 2.4, Median: 2,
Mode: [2]

```

OBSERVATION: AI generated the correct code as per I wrote in the comment, when through I wrote some irrelevant math function above the comment

The screenshot shows a Visual Studio Code editor with a Python file named `t2.py`. The code defines three functions: `calculate_average`, `calculate_median`, and `calculate_mode`. It also includes an example usage section that creates a list `data = [1, 2, 2, 3, 4]` and prints the results of the functions. The terminal at the bottom shows the output: `Average: 2.4, Median: 2, Mode: [2]`. A right-hand sidebar contains a banner that says "Build with agent mode." and "AI responses may be inaccurate."

Expected Output:

15

50

Average: 2.4, Median: 2, Mode: [2]

Task 3: Few-shot prompt

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:

```
""">>> convert_to_binary(5)
```

```
'101'
```

```
>>> convert_to_binary(10)
```

```
'1010'
```

```
>>> convert_to_binary(1)
```

```
'1'
```

```
>>> convert_to_binary(0)
```

```
'0'
```

```
>>> convert_to_binary(255)
```

```
'11111111'
```

```
"""
```

```
def convert_to_binary(n):
```

```
    if n == 0:
```

```
        return '0'
```

```
    binary = ''
```

```
    while n > 0:
```

```
        binary = str(n % 2) + binary
```

```
        n = n // 2
```

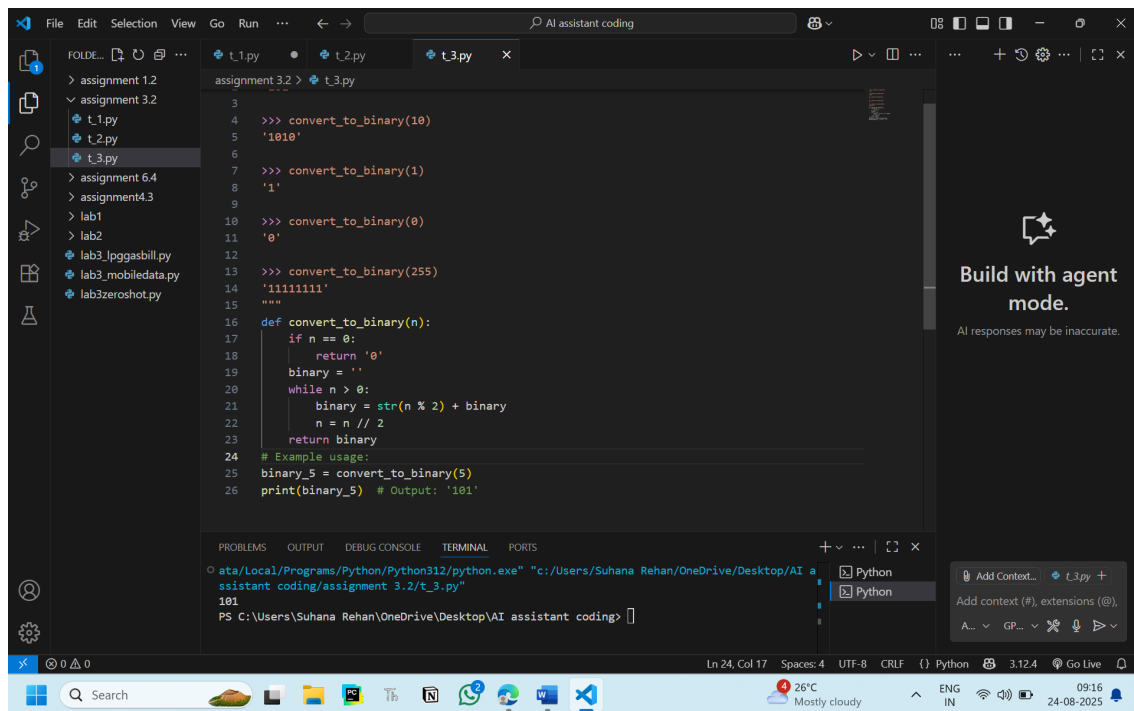
```
    return binary
```

```
# Example usage:
```

```
binary_5 = convert_to_binary(5)
```

```
print(binary_5) # Output: '101'
```

OBSERVATION: AI generates the `convert_to_binary()` function after I wrote few examples of it in the docstring



Expected Output: 101

Task 4:User Interface

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

Code:

```
import tkinter as tk
from tkinter import messagebox

class HotelBillingApp:
    def __init__(self, root):
        self.root = root
        self.root.title("Hotel Billing System")
        self.root.geometry("400x500")

        # Customer Details
        tk.Label(root, text="Customer Name:").pack()
        self.name_entry = tk.Entry(root)
        self.name_entry.pack()

        # Room Type
```

```

tk.Label(root, text="Room Type:").pack()
self.room_var = tk.StringVar(value="Standard")
room_types = ["Standard", "Deluxe", "Suite"]
for room in room_types:
    tk.Radiobutton(root, text=room, variable=self.room_var, value=room).pack(anchor='w')

# Number of Nights
tk.Label(root, text="Number of Nights:").pack()
self.nights_entry = tk.Entry(root)
self.nights_entry.pack()

# Extra Services
tk.Label(root, text="Extra Services:").pack()
self.breakfast_var = tk.IntVar()
self.laundry_var = tk.IntVar()
tk.Checkbutton(root, text="Breakfast ($10/night)", variable=self.breakfast_var).pack(anchor='w')
tk.Checkbutton(root, text="Laundry ($5/night)", variable=self.laundry_var).pack(anchor='w')

# Generate Bill Button
tk.Button(root, text="Generate Bill", command=self.generate_bill).pack(pady=10)

# Bill Output
self.bill_text = tk.Text(root, height=10, width=40)
self.bill_text.pack()

def generate_bill(self):
    name = self.name_entry.get()
    room_type = self.room_var.get()
    try:
        nights = int(self.nights_entry.get())
    except ValueError:
        messagebox.showerror("Error", "Please enter a valid number of nights.")
    return

# Room prices
prices = {"Standard": 50, "Deluxe": 80, "Suite": 120}
room_price = prices.get(room_type, 50)
room_total = room_price * nights
breakfast_total = self.breakfast_var.get() * 10 * nights
laundry_total = self.laundry_var.get() * 5 * nights
total = room_total + breakfast_total + laundry_total

bill = f"Hotel Bill\n" + "-"*30 + f"\nCustomer: {name}\nRoom Type: {room_type}\nNights:
{nights}\nRoom Charge: ${room_total}\n"
if self.breakfast_var.get():
    bill += f"Breakfast: ${breakfast_total}\n"
if self.laundry_var.get():
    bill += f"Laundry: ${laundry_total}\n"

```

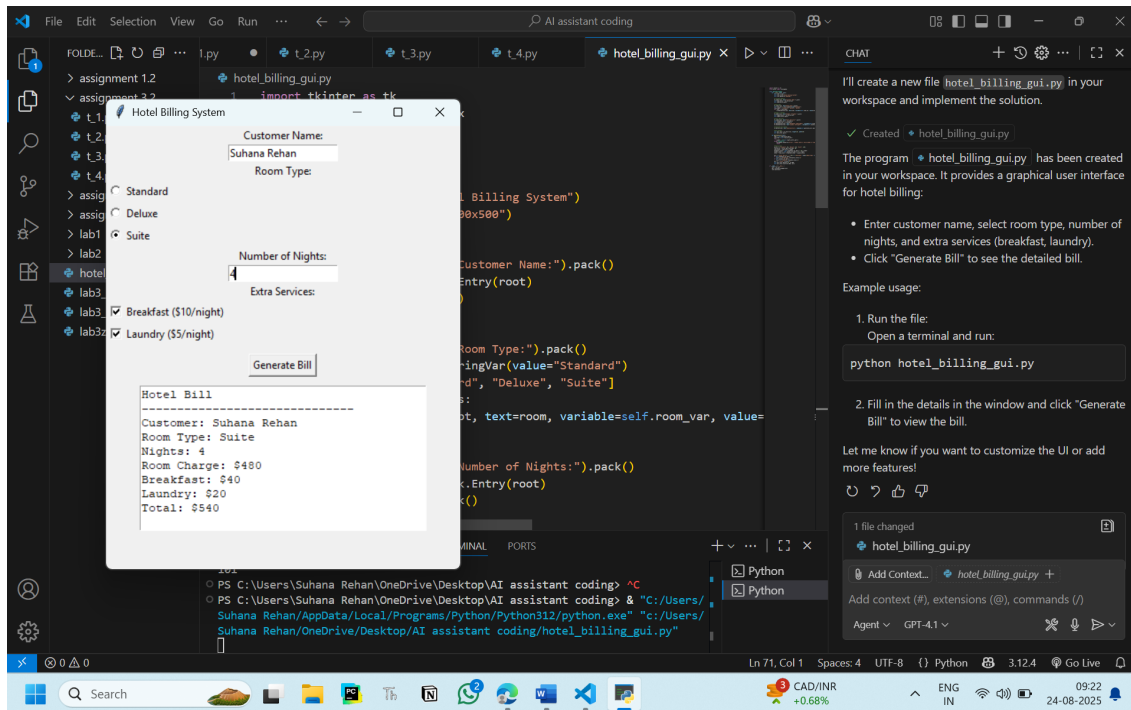
```

        bill += f"Total: ${total}\n"
        self.bill_text.delete(1.0, tk.END)
        self.bill_text.insert(tk.END, bill)

if __name__ == "__main__":
    root = tk.Tk()
    app = HotelBillingApp(root)
    root.mainloop()

```

OBSERVATION: The AI generated the user interface and functions with shared logic



Expected Output:

Hotel Bill

Customer: Suhana Rehan

Room Type: Suite

Nights: 4

Room Charge: \$480

Breakfast: \$40

Laundry: \$20

Total: \$540

Task 5: Analyzing Prompt Specificity

Prompt: Improving Temperature Conversion Function with Clear Instructions

Code:

```
# Task: Improving Temperature Conversion Function with Clear Instructions
# Write a Python function that:
# 1. Converts temperatures between Celsius and Fahrenheit.
# 2. Has a clear and descriptive function name.
# 3. Includes a docstring explaining the purpose, parameters, and return value.
# 4. Provides example input-output usage in the docstring.
# 5. Handles both "C to F" and "F to C" conversions based on an argument.
```

```
def convert_temperature(value, conversion_type):
    """
    Convert temperatures between Celsius and Fahrenheit.
```

Args:

value (float): The temperature value to be converted.
conversion_type (str): The type of conversion, either "C to F" or "F to C".

Returns:

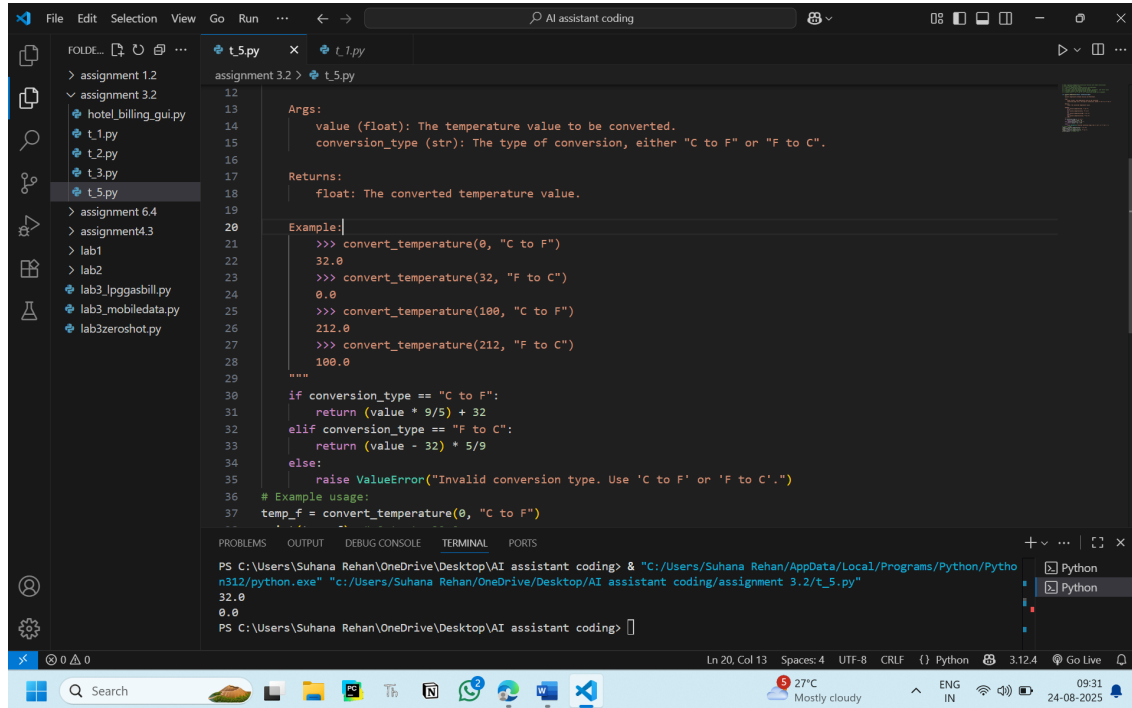
float: The converted temperature value.

Example:

```
>>> convert_temperature(0, "C to F")
32.0
>>> convert_temperature(32, "F to C")
0.0
>>> convert_temperature(100, "C to F")
212.0
>>> convert_temperature(212, "F to C")
100.0
"""
if conversion_type == "C to F":
    return (value * 9/5) + 32
elif conversion_type == "F to C":
    return (value - 32) * 5/9
else:
    raise ValueError("Invalid conversion type. Use 'C to F' or 'F to C'.")
# Example usage:
temp_f = convert_temperature(0, "C to F")
print(temp_f) # Output: 32.0
temp_c = convert_temperature(32, "F to C")
```

```
print(temp_c) # Output: 0.0
```

OBSERVATION: The AI Generates the code for Temperature Conversion Function as per the instructions given in the comment of the code



The screenshot shows a Visual Studio Code editor window with a file named `t_5.py` open. The file contains a Python function `convert_temperature` that takes a temperature value and a conversion type (either "C to F" or "F to C") and returns the converted temperature. The function includes docstrings for arguments and returns, and an example usage section. The terminal at the bottom shows the command `python t_5.py` being executed, resulting in the output `32.0` and `0.0`.

```
12
13
14 Args:
15     value (float): The temperature value to be converted.
16     conversion_type (str): The type of conversion, either "C to F" or "F to C".
17
18 Returns:
19     float: The converted temperature value.
20
21 Example:
22 >>> convert_temperature(0, "C to F")
23 32.0
24 >>> convert_temperature(32, "F to C")
25 0.0
26 >>> convert_temperature(100, "C to F")
27 212.0
28 >>> convert_temperature(212, "F to C")
29 100.0
30
31 if conversion_type == "C to F":
32     return (value * 9/5) + 32
33 elif conversion_type == "F to C":
34     return (value - 32) * 5/9
35 else:
36     raise ValueError("Invalid conversion type. Use 'C to F' or 'F to C'.")
37
38 # Example usage:
39 temp_f = convert_temperature(0, "C to F")
40
41 PS C:\Users\Suhana Rehan\OneDrive\Desktop\AI assistant coding> & "C:/Users/Suhana Rehan/AppData/Local/Programs/Python/Python312/python.exe" "C:/Users/Suhana Rehan/OneDrive/Desktop/AI assistant coding/assignment 3.2/t_5.py"
42 32.0
43 0.0
44 PS C:\Users\Suhana Rehan\OneDrive\Desktop\AI assistant coding>
```

Expected Output:

32.0

0.0