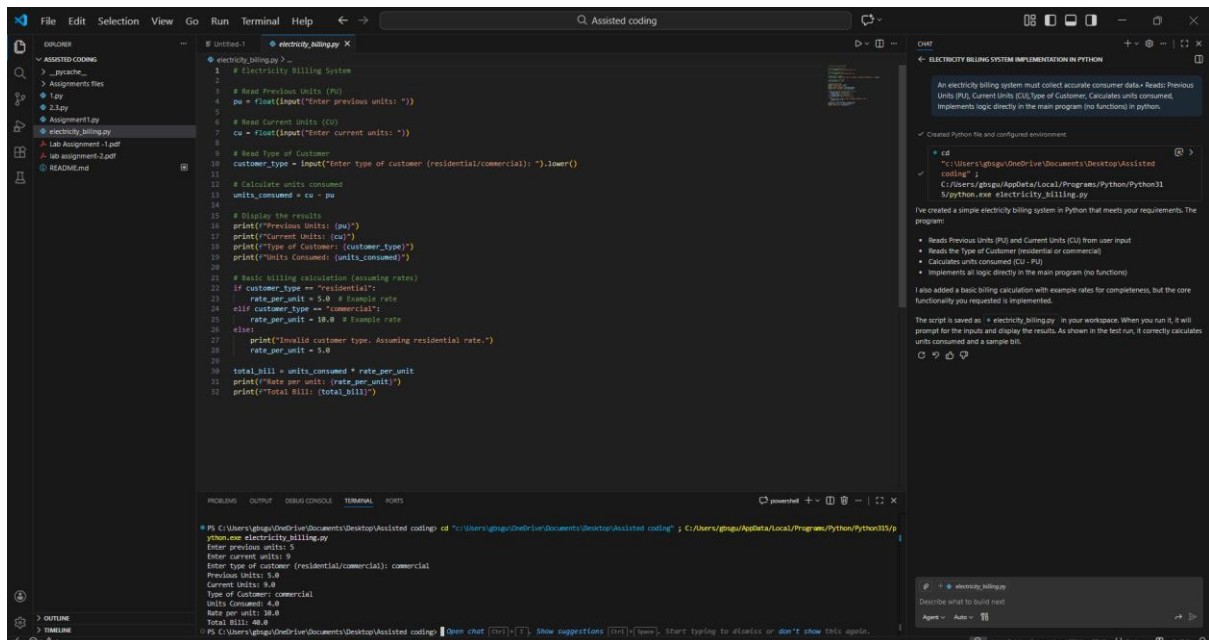


Task 1: AI-Generated Logic for Reading Consumer Details

Scenario

An electricity billing system must collect accurate consumer data.



```
# electricity_billing.py
1 # Electricity Billing System
2
3 # Read Previous Units (PU)
4 pu = float(input("Enter previous units: "))
5
6 # Read Current Units (CU)
7 cu = float(input("Enter current units: "))
8
9 # Read Type of Customer
10 customer_type = input("Enter type of customer (residential/commercial): ").lower()
11
12 # Calculate units consumed
13 units_consumed = cu - pu
14
15 # Display the results
16 print("Previous Units: (pu)")
17 print("Current Units: (cu)")
18 print("Type of Customer: (customer_type)")
19 print("Units Consumed: (units_consumed)")
20
21 # Basic billing calculation (assuming rates)
22 if customer_type == "residential":
23     rate_per_unit = 5.0 # Example rate
24 elif customer_type == "commercial":
25     rate_per_unit = 10.0 # Example rate
26 else:
27     print("Invalid customer type. Assuming residential rate.")
28     rate_per_unit = 5.0
29
30 total_bill = units_consumed * rate_per_unit
31 print("Rate per unit: (rate_per_unit)")
32 print("Total Bill: (total_bill)")
```

AI Chat Window: ELECTRICITY BILLING SYSTEM IMPLEMENTATION IN PYTHON

An electricity billing system must collect accurate consumer data. Reads: Previous Units (PU), Current Units (CU), Type of Customer. Calculates units consumed. Implements logic directly in the main program (no functions in python).

✓ Created Python file and configured environment

- Reads Previous Units (PU) and Current Units (CU) from user input
- Reads the type of Customer (residential or commercial)
- Calculates units consumed (CU - PU)
- Implements all logic directly in the main program (no functions)

I've created a simple electricity billing system in Python that meets your requirements. The program:

- Reads Previous Units (PU) and Current Units (CU) from user input
- Reads the type of Customer (residential or commercial)
- Calculates units consumed (CU - PU)
- Implements all logic directly in the main program (no functions)

I also added a basic billing calculation with example rates for completeness, but the core functionality you requested is implemented.

The script is saved as "electricity_billing.py" in your workspace. When you run it, it will prompt for the inputs and display the results. As shown in the test run, it correctly calculates units consumed and a sample bill.

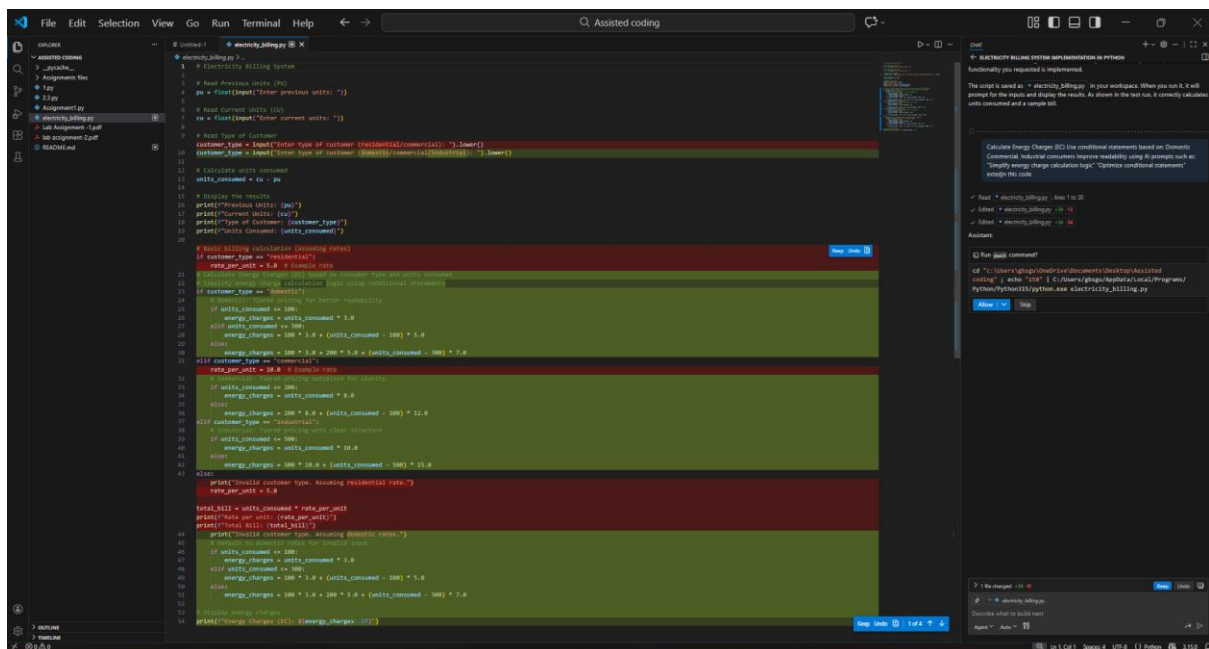
Terminal Output:

```
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & python.exe electricity_billing.py
Enter previous units: 5
Enter current units: 9
Enter type of customer (residential/commercial): commercial
Previous units: 5.0
Current units: 9.0
Type of Customer: commercial
Units Consumed: 4.0
Rate per unit: 10.0
Total Bill: 40.0
```

Task 2: Energy Charges Calculation Based on Units Consumed

Scenario

Energy charges depend on the number of units consumed and customer type.



```
# electricity_billing.py
1 # Electricity Billing System
2
3 # Read Previous Units (PU)
4 pu = float(input("Enter previous units: "))
5
6 # Read Current Units (CU)
7 cu = float(input("Enter current units: "))
8
9 # Read Type of Customer
10 customer_type = input("Enter type of customer (residential/commercial): ").lower()
11 customer_type = input("Enter type of customer (residential/commercial): ").lower()
12
13 # Calculate units consumed
14 units_consumed = cu - pu
15
16 # Display the results
17 print("Previous Units: (pu)")
18 print("Current Units: (cu)")
19 print("Type of Customer: (customer_type)")
20 print("Units Consumed: (units_consumed)")
21
22 # Basic billing calculation (assuming rates)
23 if customer_type == "residential":
24     rate_per_unit = 5.0 # Example rate
25 elif customer_type == "commercial":
26     rate_per_unit = 10.0 # Example rate
27 else:
28     print("Invalid customer type. Assuming residential rate.")
29     rate_per_unit = 5.0
30
31 total_bill = units_consumed * rate_per_unit
32 print("Rate per unit: (rate_per_unit)")
33 print("Total Bill: (total_bill)")
34
35 # Calculate energy charges based on units consumed
36 if units_consumed <= 100:
37     energy_charges = units_consumed * 5.0
38 elif units_consumed > 100:
39     energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
40 else:
41     energy_charges = 100 * 5.0 + 200 * 5.0 + (units_consumed - 300) * 7.0
42
43 if customer_type == "commercial":
44     rate_per_unit = 10.0 # Example rate
45 else:
46     print("Invalid customer type. Assuming residential rate.")
47     rate_per_unit = 5.0
48
49 total_bill = units_consumed * rate_per_unit
50 print("Rate per unit: (rate_per_unit)")
51 print("Total Bill: (total_bill)")
52
53 # Calculate energy charges based on units consumed
54 if units_consumed <= 100:
55     energy_charges = units_consumed * 5.0
56 elif units_consumed > 100:
57     energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
58 else:
59     energy_charges = 100 * 5.0 + 200 * 5.0 + (units_consumed - 300) * 7.0
60
61 # Display energy charges
62 print("Energy charges (EC): (energy_charges)")
```

AI Chat Window: ELECTRICITY BILLING SYSTEM IMPLEMENTATION IN PYTHON

The code is used as "electricity_billing.py" in your workspace. When you run it, it will prompt for the inputs and display the results. As shown in the test run, it correctly calculates units consumed and a sample bill.

✓ Read "electricity_billing.py" from workspace

✓ Edited "electricity_billing.py" to add energy charge calculation logic. "Optimize conditional statements" inside the code.

Assistant:

[] Run (bash) command

```
cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding" & python.exe "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding\electricity_billing.py"
```

Terminal Output:

```
1 # Electricity Billing System
2 # Read Previous Units (pu)
3 pu = float(input("Enter previous units: "))
4
5 # Read Current Units (cu)
6 cu = float(input("Enter current units: "))
7
8 # Read type of customer
9 customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()
10
11 # Calculate units consumed
12 units_consumed = cu - pu
13
14 # Display the results
15 print(f"Previous Units: {pu}")
16 print(f"Current Units: {cu}")
17 print(f"Type of Customer: {customer_type}")
18 print(f"Units Consumed: {units_consumed}")
19
20 # Calculate Energy Charges (EC) based on customer type and units consumed
21 # Handle energy charge calculation logic using conditional statements
22 if customer_type == "domestic":
23     # Domestic tiered pricing for better readability
24     if units_consumed <= 100:
25         energy_charges = units_consumed * 5.0
26     elif units_consumed <= 200:
27         energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
28     else:
29         energy_charges = 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
30 elif customer_type == "commercial":
31     # Commercial tiered pricing (simplified for clarity)
32     if units_consumed <= 200:
33         energy_charges = units_consumed * 8.0
34     else:
35         energy_charges = 200 * 8.0 + (units_consumed - 200) * 12.0
36 elif customer_type == "industrial":
37     # Industrial tiered pricing with clear structure
38     if units_consumed <= 500:
39         energy_charges = units_consumed * 10.0
40     else:
41         energy_charges = 500 * 10.0 + (units_consumed - 500) * 15.0
42 else:
43     print("Invalid customer type. Assuming Domestic rates.")
44 # Default to domestic rates for invalid input
45 if units_consumed <= 100:
46     energy_charges = units_consumed * 5.0
47 elif units_consumed <= 200:
48     energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
49 else:
50     energy_charges = 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
51
52 # Display energy charges
53 print(f"Energy Charges (EC): ${energy_charges:.2f}")
54
55 # Main execution
56 calculate_energy_charges(pu, cu, customer_type)
```

PS C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding> python.exe "C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding\electricity_billing.py"

Enter previous units: 0
Enter current units: 4
Enter type of customer (domestic/commercial/industrial): Industrial
Previous Units: 0.0
Current Units: 4.0
Type of Customer: Industrial
Units Consumed: 4.0
Energy Charges (EC): \$20.00
PS C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding>

Task 3: Modular Design Using AI Assistance (Using Functions)

Scenario

Billing logic must be reusable for multiple consumers.

```
1 # Electricity Billing System with User-Defined Functions
2 # This program calculates electricity bills using modular functions for energy charges and fixed charges
3
4 def calculate_energy_charges(units_consumed, customer_type):
5     """
6     Calculate energy charges based on customer type and units consumed.
7     Uses tiered pricing structure for different consumption levels.
8     """
9     Args:
10         units_consumed (float): Number of units consumed
11         customer_type (str): Type of customer ('domestic', 'commercial', 'industrial')
12
13     Returns:
14         float: Calculated energy charges
15
16     """
17     if customer_type == "domestic":
18         # Domestic tiered pricing: 5.0 for first 100, 7.0 for next 200, 7.0 thereafter
19         if units_consumed <= 100:
20             return units_consumed * 5.0
21         elif units_consumed <= 200:
22             return 100 * 5.0 + (units_consumed - 100) * 7.0
23         else:
24             return 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
25     elif customer_type == "commercial":
26         # Commercial tiered pricing: 8.0 for first 200, 12.0 thereafter
27         if units_consumed <= 200:
28             return units_consumed * 8.0
29         else:
30             return 200 * 8.0 + (units_consumed - 200) * 12.0
31     elif customer_type == "industrial":
32         # Industrial tiered pricing: 10.0 for first 500, 15.0 thereafter
33         if units_consumed <= 500:
34             return units_consumed * 10.0
35         else:
36             return 500 * 10.0 + (units_consumed - 500) * 15.0
37     else:
38         # Default to domestic rates for invalid customer type
39         print("Invalid customer type. Using domestic rates.")
40         if units_consumed <= 100:
41             return units_consumed * 5.0
42         elif units_consumed <= 200:
43             return 100 * 5.0 + (units_consumed - 100) * 7.0
44         else:
45             return 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
46
47 def calculate_fixed_charges(customer_type):
48     """
49     Calculate fixed charges based on customer type.
50     """
51     Args:
52         customer_type (str): Type of customer ('domestic', 'commercial', 'industrial')
53
54     Returns:
55         float: Calculated fixed charges
56
57     """
58     if customer_type == "domestic":
59         return 10.0
60     elif customer_type == "commercial":
61         return 20.0
62     elif customer_type == "industrial":
63         return 50.0
64     else:
65         return 10.0
66
67 # Main function to handle user input, call the calculation functions,
68 # display results, and return the calculated values as a tuple.
69 def main():
70     pu = float(input("Enter previous units: "))
71     cu = float(input("Enter current units: "))
72     customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()
73
74     units_consumed = cu - pu
75
76     energy_charges = calculate_energy_charges(units_consumed, customer_type)
77     fixed_charges = calculate_fixed_charges(customer_type)
78
79     total_bill = energy_charges + fixed_charges
80
81     print(f"Electricity Bill Summary")
82     print(f"Previous Units: {pu}")
83     print(f"Current Units: {cu}")
84     print(f"Units Consumed: {units_consumed}")
85     print(f"Customer Type: {customer_type}")
86     print(f"Energy Charges: ${energy_charges:.2f}")
87     print(f"Fixed Charges: ${fixed_charges:.2f}")
88     print(f"Total Bill: ${total_bill:.2f}")
89
90 if __name__ == "__main__":
91     main()
```

PS C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding> python.exe "C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding\33.py"

Enter previous units: 0
Enter current units: 4
Enter type of customer (domestic/commercial/industrial): Industrial
Electricity Bill Summary
Previous Units: 0.0
Current Units: 4.0
Units Consumed: 4.0
Customer Type: Industrial
Energy Charges: \$20.00
Fixed Charges: \$50.00
Total Bill: \$70.00
PS C:\Users\ghgu\OneDrive\Documents\Desktop\Assisted coding>

```
def calculate_energy_charges(units_consumed, customer_type):
    """Calculate energy charges based on units consumed and customer type.
    Returns a tuple of (energy_charges, fixed_charges)."""
    if units_consumed <= 100:
        return 100 * 3.0 + (units_consumed - 100) * 5.0
    else:
        return 100 * 3.0 + 200 * 5.0 + (units_consumed - 300) * 7.0

def calculate_fixed_charges(customer_type):
    """Calculate fixed charges based on customer type.
    Returns a tuple of (fixed_charges, service_fee)."""
    if customer_type == "domestic":
        return 50.0 # Monthly service fee for domestic customers
    elif customer_type == "commercial":
        return 100.0 # Monthly service fee for commercial customers
    elif customer_type == "industrial":
        return 200.0 # Monthly service fee for industrial customers
    else:
        # Default to domestic fixed charges
        return 50.0

def main():
    """Main function to handle user input and display billing information.
    Calls the calculation functions and returns the calculated values.
    """
    # Read Previous Units (PU)
    pu = float(input("Enter previous units: "))

    # Read Current Units (CU)
    cu = float(input("Enter current units: "))

    # Read Type of Customer
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

    # Calculate energy charges
    energy_charges, fixed_charges = calculate_energy_charges(cu - pu, customer_type)

    # Calculate fixed charges
    fixed_charges, service_fee = calculate_fixed_charges(customer_type)

    # Calculate total bill
    total_bill = energy_charges + fixed_charges + service_fee

    # Display the results
    print("\n--- Electricity Bill ---")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Units Consumed: {cu - pu}")
    print(f"Customer Type: {customer_type}")
    print(f"Energy Charges: ${energy_charges:.2f}")
    print(f"Fixed Charges: ${fixed_charges:.2f}")
    print(f"Total Bill: ${total_bill:.2f}")

    # Return calculated values as a tuple
    return energy_charges, fixed_charges, total_bill

# Run the main function if this script is executed directly
if __name__ == "__main__":
    main()
```

Terminal Output:

```
PS C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding> C:\Users\gmgpu\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding\3.3.py"

--- Electricity Bill ---
Previous Units: 10.0
Current Units: 12.0
Units Consumed: 2.0
Customer Type: Commercial
Energy Charges: $16.00
Fixed Charges: $100.00
Total Bill: $116.00
```

```
def main():
    """Main function to handle user input and display billing information.
    Calls the calculation functions and returns the calculated values.
    """
    # Read Previous Units (PU)
    pu = float(input("Enter previous units: "))

    # Read Current Units (CU)
    cu = float(input("Enter current units: "))

    # Read Type of Customer
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

    # Calculate energy charges
    energy_charges, fixed_charges = calculate_energy_charges(cu - pu, customer_type)

    # Calculate fixed charges
    fixed_charges, service_fee = calculate_fixed_charges(customer_type)

    # Calculate total bill
    total_bill = energy_charges + fixed_charges + service_fee

    # Display the results
    print("\n--- Electricity Bill ---")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Units Consumed: {cu - pu}")
    print(f"Customer Type: {customer_type}")
    print(f"Energy Charges: ${energy_charges:.2f}")
    print(f"Fixed Charges: ${fixed_charges:.2f}")
    print(f"Total Bill: ${total_bill:.2f}")

    # Return calculated values as a tuple
    return energy_charges, fixed_charges, total_bill

# Run the main function if this script is executed directly
if __name__ == "__main__":
    main()
```

Terminal Output:

```
PS C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding> C:\Users\gmgpu\AppData\Local\Programs\Python\Python311\python.exe "C:\Users\gmgpu\OneDrive\Documents\Desktop\Assisted coding\3.3.py"

--- Electricity Bill ---
Previous Units: 10.0
Current Units: 12.0
Units Consumed: 2.0
Customer Type: Commercial
Energy Charges: $16.00
Fixed Charges: $100.00
Total Bill: $116.00
```

Task 4: Calculation of Additional Charges

Scenario

Electricity bills include multiple additional charges.

```
File Edit Selection View Go Run Terminal Help
electric_billings.py
# Electric Billings System - Enhanced Calculations
# Read previous units (pu)
pu = float(input("Enter previous units: "))

# Read current units (cu)
cu = float(input("Enter current units: "))

# Read type of customer
customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

# Calculate units consumed
units_consumed = cu - pu

# Display the results
print("Previous units: {pu}")
print("Current units: {cu}")
print("Type of customer: {customer_type}")
print("Units Consumed: {units_consumed}")

# Calculate Energy Charges (EC) based on customer type and units consumed
# Simplify energy charge calculation logic using conditional statements
if customer_type == "domestic":
    # Domestic: tiered pricing for better readability
    if units_consumed <= 100:
        energy_charges = units_consumed * 3.0
    elif units_consumed > 100:
        energy_charges = 100 * 3.0 + 3.0 * (units_consumed - 100) * 5.0
    else:
        energy_charges = 100 * 3.0 + 200 * 5.0 * (units_consumed - 100) * 7.0
elif customer_type == "commercial":
    # Commercial: tiered pricing for better readability
    if units_consumed <= 100:
        energy_charges = units_consumed * 5.0
    elif units_consumed > 100:
        energy_charges = 100 * 5.0 + 5.0 * (units_consumed - 100) * 12.0
elif customer_type == "industrial":
    # Industrial: tiered pricing with clear structure
    if units_consumed <= 100:
        energy_charges = units_consumed * 10.0
    elif units_consumed > 100:
        energy_charges = 100 * 10.0 + 10.0 * (units_consumed - 100) * 15.0
else:
    print("Invalid customer type. Assuming domestic rates.")
    # Default to domestic rates for invalid input
    if units_consumed <= 100:
        energy_charges = units_consumed * 3.0
    elif units_consumed > 100:
        energy_charges = 100 * 3.0 + 3.0 * (units_consumed - 100) * 5.0

# Calculate Fixed Charges (FC) based on customer type
if customer_type == "domestic":
    fixed_charges = 50.0 # Monthly service fee for domestic customers
elif customer_type == "commercial":
    fixed_charges = 100.0 # Monthly service fee for commercial customers
elif customer_type == "industrial":
    fixed_charges = 200.0 # Monthly service fee for industrial customers
else:
    fixed_charges = 50.0 # Default to domestic fixed charges

# Calculate Electricity Duty (ED) as percentage of energy charges (EC)
# Electricity duty rate varies by customer type for improved billing accuracy
if customer_type == "domestic":
    electricity_duty_rate = 0.05 # 5% for domestic
elif customer_type == "commercial":
    electricity_duty_rate = 0.08 # 8% for commercial
elif customer_type == "industrial":
    electricity_duty_rate = 0.10 # 10% for industrial
else:
    electricity_duty_rate = 0.05 # Default 5%

electricity_duty = energy_charges * electricity_duty_rate

# Calculate Customer Charges (CC) - Total charges including all components
customer_charges = energy_charges + fixed_charges + electricity_duty

# Display the results
print("----- Electricity Bill Details -----")
print(f"Energy Charges (EC): {energy_charges:.2f}")
print(f"Fixed Charges (FC): {fixed_charges:.2f}")
print(f"Electricity Duty (ED): {electricity_duty:.2f} ({electricity_duty_rate*100:.0f}%)")
print(f"Customer Charges (CC): {customer_charges:.2f}")

# Run python command?
python "C:\Users\gghugh\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\gghugh\Documents\Desktop\Assisted coding\electricity_billings.py"
```

```
File Edit Selection View Go Run Terminal Help
electric_billings.py
# Electric Billings System - Enhanced Calculations
# Read previous units (pu)
pu = float(input("Enter previous units: "))

# Read current units (cu)
cu = float(input("Enter current units: "))

# Read type of customer
customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()

# Calculate units consumed
units_consumed = cu - pu

# Display the results
print("Previous units: {pu}")
print("Current units: {cu}")
print("Type of customer: {customer_type}")
print("Units Consumed: {units_consumed}")

# Calculate Energy Charges (EC) based on customer type and units consumed
# Simplify energy charge calculation logic using conditional statements
if customer_type == "domestic":
    # Domestic: tiered pricing for better readability
    if units_consumed <= 100:
        energy_charges = units_consumed * 3.0
    elif units_consumed > 100:
        energy_charges = 100 * 3.0 + 3.0 * (units_consumed - 100) * 5.0
    else:
        energy_charges = 100 * 3.0 + 200 * 5.0 * (units_consumed - 100) * 7.0
elif customer_type == "commercial":
    # Commercial: tiered pricing for better readability
    if units_consumed <= 100:
        energy_charges = units_consumed * 5.0
    elif units_consumed > 100:
        energy_charges = 100 * 5.0 + 5.0 * (units_consumed - 100) * 12.0
elif customer_type == "industrial":
    # Industrial: tiered pricing with clear structure
    if units_consumed <= 100:
        energy_charges = units_consumed * 10.0
    elif units_consumed > 100:
        energy_charges = 100 * 10.0 + 10.0 * (units_consumed - 100) * 15.0
else:
    print("Invalid customer type. Assuming domestic rates.")
    # Default to domestic rates for invalid input
    if units_consumed <= 100:
        energy_charges = units_consumed * 3.0
    elif units_consumed > 100:
        energy_charges = 100 * 3.0 + 3.0 * (units_consumed - 100) * 5.0

# Calculate Fixed Charges (FC) based on customer type
if customer_type == "domestic":
    fixed_charges = 50.0 # Monthly service fee for domestic customers
elif customer_type == "commercial":
    fixed_charges = 100.0 # Monthly service fee for commercial customers
elif customer_type == "industrial":
    fixed_charges = 200.0 # Monthly service fee for industrial customers
else:
    fixed_charges = 50.0 # Default to domestic fixed charges

# Calculate Electricity Duty (ED) as percentage of energy charges (EC)
# Electricity duty rate varies by customer type for improved billing accuracy
if customer_type == "domestic":
    electricity_duty_rate = 0.05 # 5% for domestic
elif customer_type == "commercial":
    electricity_duty_rate = 0.08 # 8% for commercial
elif customer_type == "industrial":
    electricity_duty_rate = 0.10 # 10% for industrial
else:
    electricity_duty_rate = 0.05 # Default 5%

electricity_duty = energy_charges * electricity_duty_rate

# Calculate Customer Charges (CC) - Total charges including all components
customer_charges = energy_charges + fixed_charges + electricity_duty

# Display the results
print("----- Electricity Bill Details -----")
print(f"Energy Charges (EC): {energy_charges:.2f}")
print(f"Fixed Charges (FC): {fixed_charges:.2f}")
print(f"Electricity Duty (ED): {electricity_duty:.2f} ({electricity_duty_rate*100:.0f}%)")
print(f"Customer Charges (CC): {customer_charges:.2f}")

# Run python command?
python "C:\Users\gghugh\AppData\Local\Programs\Python\Python311\python.exe" "C:\Users\gghugh\Documents\Desktop\Assisted coding\electricity_billings.py"
```

Task 5: Final Bill Generation and Output Analysis

Scenario

The final electricity bill must present all values clearly.

```
File Edit Selection View Go Run Terminal Help
electricity_bill.py
# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components.
# It includes validation and error handling for real-world applicability.

try:
    # Read previous units (Only with validation)
    prev = float(input("Enter previous units: "))
    if prev < 0:
        raise ValueError("Previous units cannot be negative")

    # Read current units (Only with validation)
    curr = float(input("Enter current units: "))
    if curr < 0:
        raise ValueError("Current units cannot be negative")

    # Read type of customer with validation
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower().strip()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        customer_type = "domestic" # Default fallback
        print("Using default customer type: {customer_type}")

except ValueError as e:
    print(f"❌ Input Error: {e}")
    print("Please enter valid numeric values for units.")
    exit()

# Calculate units consumed with validation
units_consumed = curr - prev

# Handle negative consumption (possible meter reset or error)
if units_consumed < 0:
    print(f"⚠️ Warning: Current units are less than previous units.")
    print("This may indicate a meter reading error or meter reset.")
    print("Setting energy charges to $0.00 for this billing cycle.")
    energy_charges = 0.0
    electricity_duty = 0.0
    units_consumed_display = f"({units_consumed:1f}) (adjusted to 0 for billing)"
else:
    units_consumed_display = f"{units_consumed:1f}"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Standard pricing structure for accurate billing
    if customer_type == "domestic":
        # Domestic tiered pricing: 0.0 for first 100, 0.0 for next 200, 7.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0
    elif customer_type == "commercial":
        # Commercial tiered pricing: 0.0 for first 200, 12.0 thereafter
        if units_consumed <= 200:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 200 * 0.0 + (units_consumed - 200) * 12.0
    elif customer_type == "industrial":
        # Industrial tiered pricing: 10.0 for first 100, 15.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 100 * 10.0 + (units_consumed - 100) * 15.0
    else:
        # Default to domestic rates for invalid input
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0

    # Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
    if customer_type == "domestic":
        electricity_duty_rate = 0.05 # 5% for domestic
    elif customer_type == "commercial":
        electricity_duty_rate = 0.08 # 8% for commercial
    elif customer_type == "industrial":
        electricity_duty_rate = 0.10 # 10% for industrial

    electricity_duty = 0.05 * energy_charges # 5% for domestic

# Meter Reading Summary ---
prev_units = prev
curr_units = curr
customer_type = customer_type
units_consumed = units_consumed

--- Electricity Bill Details ---
Energy Charges (EC): $0.00
Fixed Charges (FC): $0.00
Customer Charges (CC): $0.00
Electricity Duty (ED): $0.00 (5%)
Total Bill Amount: $0.00

--- Bill Summary for Industrial Customer ---
Meter Structure: Tiered pricing applied
```

```
electricity_bill.py
# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components.
# It includes validation and error handling for real-world applicability.

try:
    # Read previous units (Only with validation)
    prev = float(input("Enter previous units: "))
    if prev < 0:
        raise ValueError("Previous units cannot be negative")

    # Read current units (Only with validation)
    curr = float(input("Enter current units: "))
    if curr < 0:
        raise ValueError("Current units cannot be negative")

    # Read type of customer with validation
    customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower().strip()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        customer_type = "domestic" # Default fallback
        print("Using default customer type: {customer_type}")

except ValueError as e:
    print(f"❌ Input Error: {e}")
    print("Please enter valid numeric values for units.")
    exit()

# Calculate units consumed with validation
units_consumed = curr - prev

# Handle negative consumption (possible meter reset or error)
if units_consumed < 0:
    print(f"⚠️ Warning: Current units are less than previous units.")
    print("This may indicate a meter reading error or meter reset.")
    print("Setting energy charges to $0.00 for this billing cycle.")
    energy_charges = 0.0
    electricity_duty = 0.0
    units_consumed_display = f"({units_consumed:1f}) (adjusted to 0 for billing)"
else:
    units_consumed_display = f"{units_consumed:1f}"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Standard pricing structure for accurate billing
    if customer_type == "domestic":
        # Domestic tiered pricing: 0.0 for first 100, 0.0 for next 200, 7.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0
    elif customer_type == "commercial":
        # Commercial tiered pricing: 0.0 for first 200, 12.0 thereafter
        if units_consumed <= 200:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 200 * 0.0 + (units_consumed - 200) * 12.0
    elif customer_type == "industrial":
        # Industrial tiered pricing: 10.0 for first 100, 15.0 thereafter
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        else:
            energy_charges = 100 * 10.0 + (units_consumed - 100) * 15.0
    else:
        # Default to domestic rates for invalid input
        if units_consumed <= 100:
            energy_charges = units_consumed * 0.0
        elif units_consumed <= 300:
            energy_charges = 100 * 0.0 + (units_consumed - 100) * 0.0
        else:
            energy_charges = 100 * 0.0 + 200 * 0.0 + (units_consumed - 300) * 7.0

    # Calculate Electricity Duty (ED) as percentage of Energy Charges (EC)
    if customer_type == "domestic":
        electricity_duty_rate = 0.05 # 5% for domestic
    elif customer_type == "commercial":
        electricity_duty_rate = 0.08 # 8% for commercial
    elif customer_type == "industrial":
        electricity_duty_rate = 0.10 # 10% for industrial

    electricity_duty = 0.05 * energy_charges # 5% for domestic

# Meter Reading Summary ---
prev_units = prev
curr_units = curr
customer_type = customer_type
units_consumed = units_consumed

--- Electricity Bill Details ---
Energy Charges (EC): $0.00
Fixed Charges (FC): $0.00
Customer Charges (CC): $0.00
Electricity Duty (ED): $0.00 (5%)
Total Bill Amount: $0.00

--- Bill Summary for Industrial Customer ---
Meter Structure: Tiered pricing applied
```

```
File Edit Selection View Go Run Terminal Help ← → Q Assisted coding

# electricity_billings.py
1 # electricity_billings.py
2 #
3 # if customer_type == "residential":
4 #     customer_charges = 15.0
5 # elif customer_type == "commercial":
6 #     customer_charges = 25.0
7 # elif customer_type == "industrial":
8 #     customer_charges = 40.0
9 # else:
10 #     customer_charges = 0.0
11
12 # calculate total bill amount
13 total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
14
15 # Display the input summary
16
17 print("\n--- Meter Reading Summary ---")
18 print(f"Previous units: {prev_units}")
19 print(f"Current units: {curr_units}")
20 print(f"Customer Type: {customer_type.title()}")
21 print(f"Units Consumed: {units_consumed_display}")
22
23 # Display all charges with improved formatting for billing accuracy
24
25 print("\n--- Electricity Bill Details ---")
26 print(f"Energy Charges (EC): {energy_charges:.2f}")
27 print(f"Fixed Charges (FC): {fixed_charges:.2f}")
28 print(f"Customer Charges (CC): {customer_charges:.2f}")
29 print(f"Electricity Duty (ED): {electricity_duty:.2f} ({electricity_duty_rate*100:.0f}% of energy charges)")
30 print(f"Total Bill Amount: {total_bill:.2f}")
31
32 # Real-world applicability notes
33 print("\n--- Bill Summary for {customer_type.title()} Customer ---")
34 print(f"Rate structure: {rate_pricing_display}")
35 if units_consumed < 0:
36     print(f"Error: {electricity_duty_rate*100:.0f}% of energy charges")
37 else:
38     print(f"Units consumed: {units_consumed}")
39
40 # Customer billing accuracy checks
41 if units_consumed < 0:
42     print(f"Warning: Current units are less than previous units. Please verify meter readings.")
43 if energy_charges < 0:
44     print(f"Error: Negative energy charges detected. Check input values.")
45
46
47 # Main execution
48 prev_units = 15
49 curr_units = 25
50 customer_type = "Industrial"
51
52 # --- Meter Reading Summary ---
53 Previous units: 15.0
54 Current units: 25.0
55 Customer Type: Industrial
56 Units Consumed: 10.0
57
58 --- Electricity Bill Details ---
59 Energy Charges (EC): $30.00
60 Fixed Charges (FC): $10.00
61 Customer Charges (CC): $40.00
62 Electricity Duty (ED): $1.00 (10% of energy charges)
63 Total Bill Amount: $81.00
64
65 --- Bill Summary for Industrial Customer ---
66 Rate structure: Tiered pricing applied
```

This program accurately calculates the electricity bill by using basic arithmetic formulas. The code is easy to read because of meaningful variable names and clear print statements. It is applicable in real-world situations as it follows the standard electricity billing structure used by power departments. The formatted output helps users understand each charge clearly.

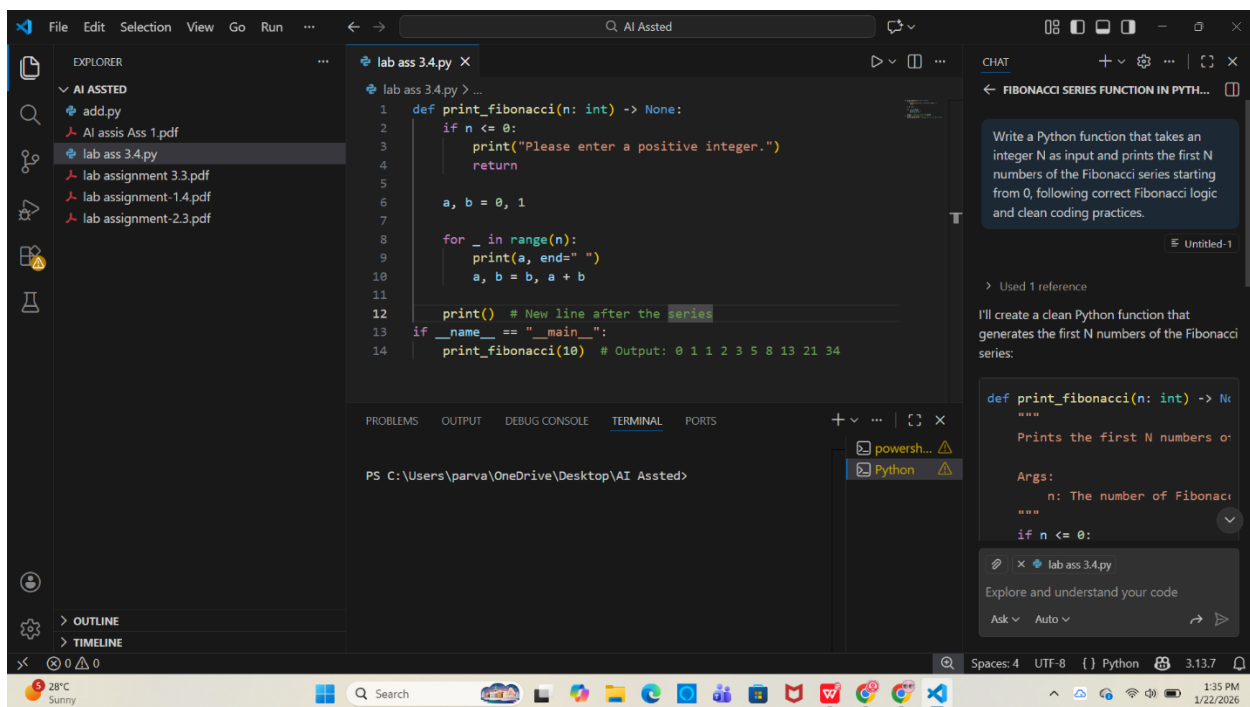
Task 1: Zero-shot Prompt – Fibonacci Series Generator

Task Description #1

- Without giving an example, write a single comment prompt asking GitHub Copilot to generate a Python function to print the first N Fibonacci numbers.

Expected Output #1

- A complete Python function generated by Copilot without any example provided.
- Correct output for sample input N = 7 → 0 1 1 2 3 5 8
- Observation on how Copilot understood the instruction with zero context.



Task 2: One-shot Prompt – List Reversal Function

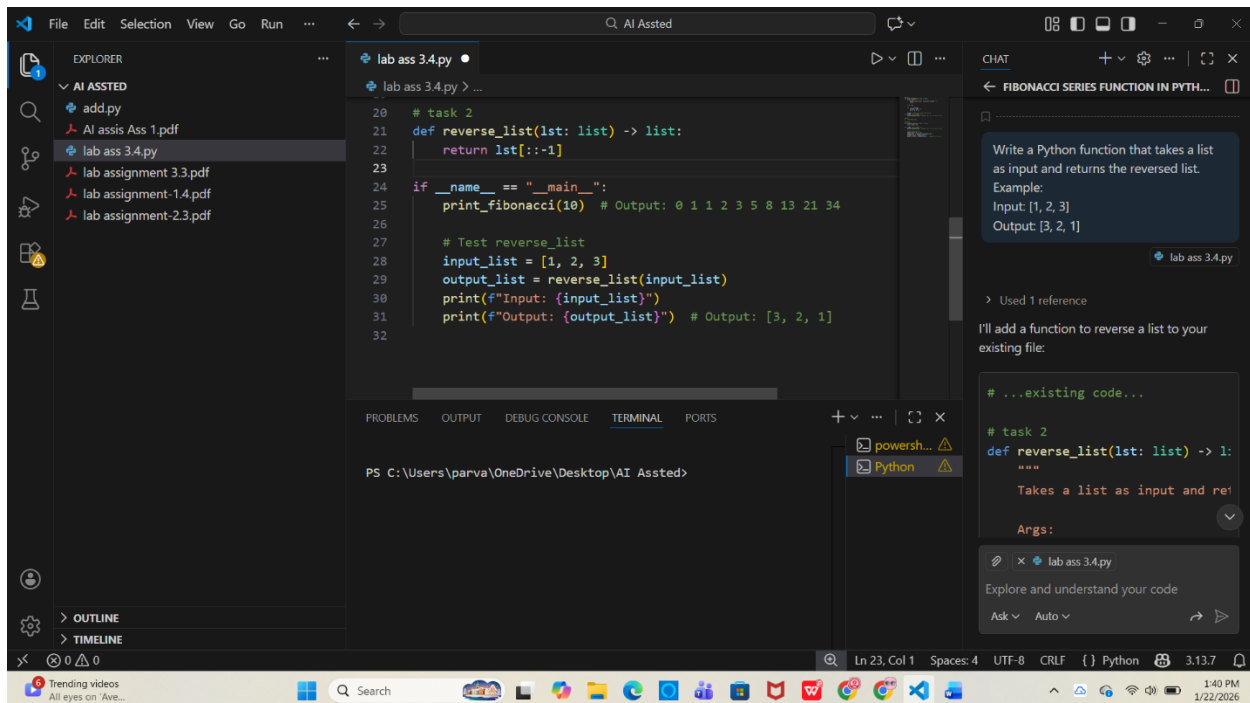
Task Description #2

- Write a comment prompt to reverse a list and provide one example below the comment to guide Copilot.

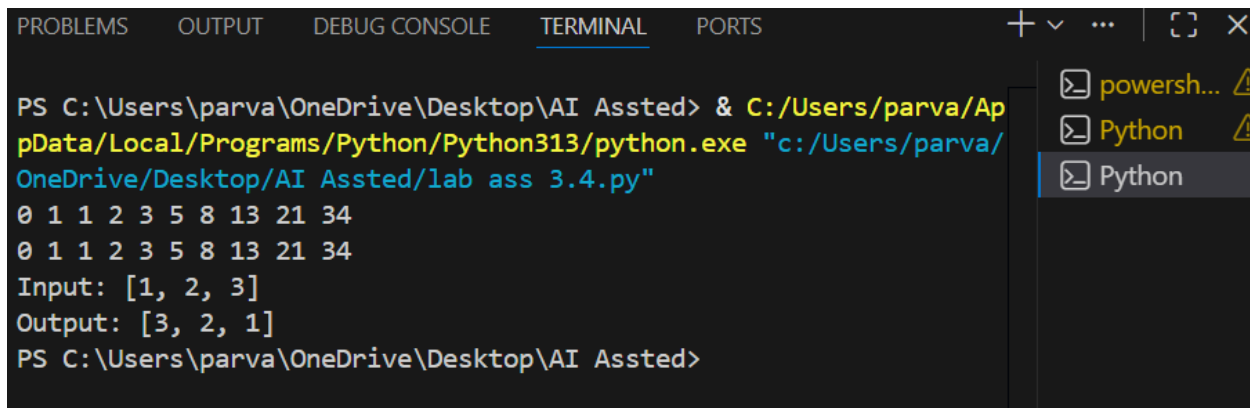
Expected Output #2

- Copilot-generated function to reverse a list using slicing or loop.
- Output: [3, 2, 1] for input [1, 2, 3]
- Observation on how adding a single example improved Copilot's

Accuracy



1 and 2 tasks output



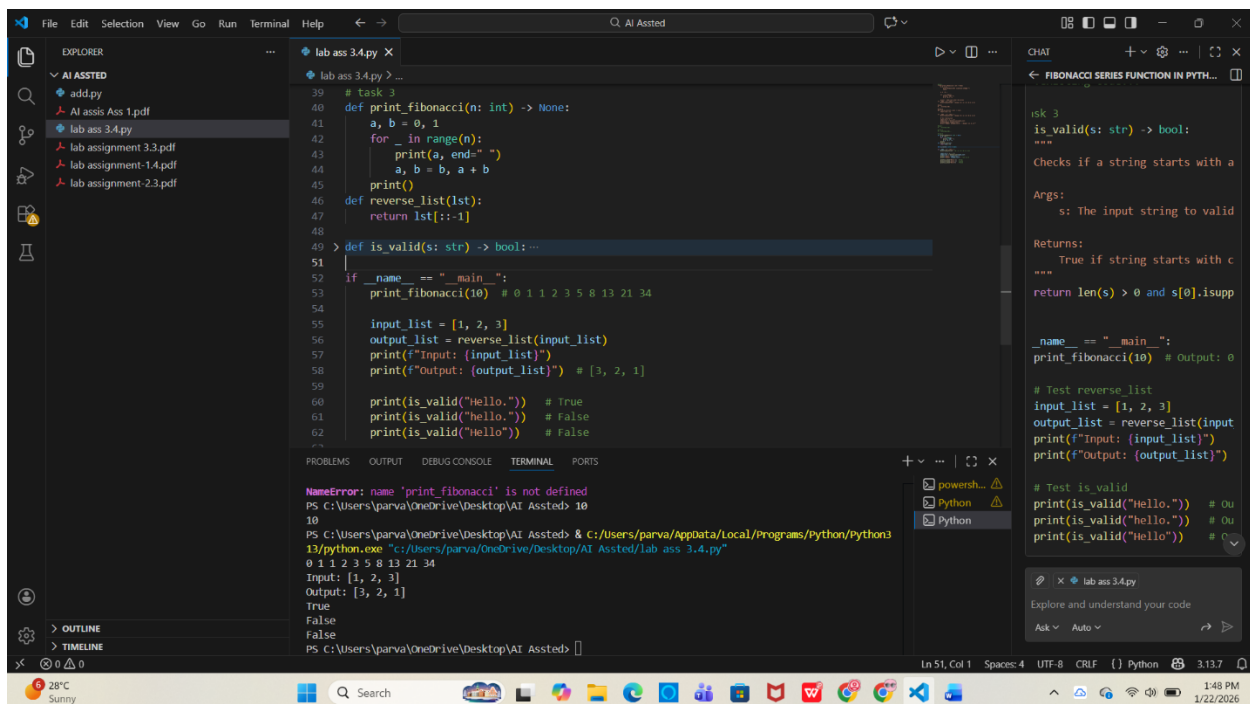
Task 3: Few-shot Prompt – String Pattern Matching

Task Description #3

- Write a comment with 2–3 examples to help Copilot understand how to check if a string starts with a capital letter and ends with a period.

Expected Output #3

- A function `is_valid()` that checks the pattern.
- Output: True or False based on input.
- Students reflect on how multiple examples guide Copilot to generate more accurate code.



```
# Task 3
def print_fibonacci(n: int) -> None:
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b
    print()

def reverse_list(lst):
    return lst[::-1]

def is_valid(s: str) -> bool:
    """
    Checks if a string starts with a capital letter and ends with a period.
    Args:
        s: The input string to validate
    Returns:
        True if string starts with c
    """
    return len(s) > 0 and s[0].isupper() and s[-1] == '.'

if __name__ == "__main__":
    print_fibonacci(10) # 0 1 1 2 3 5 8 13 21 34

    input_list = [1, 2, 3]
    output_list = reverse_list(input_list)
    print(f"Input: {input_list}")
    print(f"Output: {output_list}") # [3, 2, 1]

    print(is_valid("Hello.")) # True
    print(is_valid("hello.")) # False
    print(is_valid("Hello")) # False
```

Terminal Output:

```
PS C:\Users\parva\OneDrive\Desktop\AI Assted> 10
0 1 1 2 3 5 8 13 21 34
Input: [1, 2, 3]
Output: [3, 2, 1]
True
False
False
```

Task 4: Zero-shot vs Few-shot – Email Validator

Task Description #4

- First, prompt Copilot to write an email validation function using zero-shot (just the task in comment).
- Then, rewrite the prompt using few-shot examples.

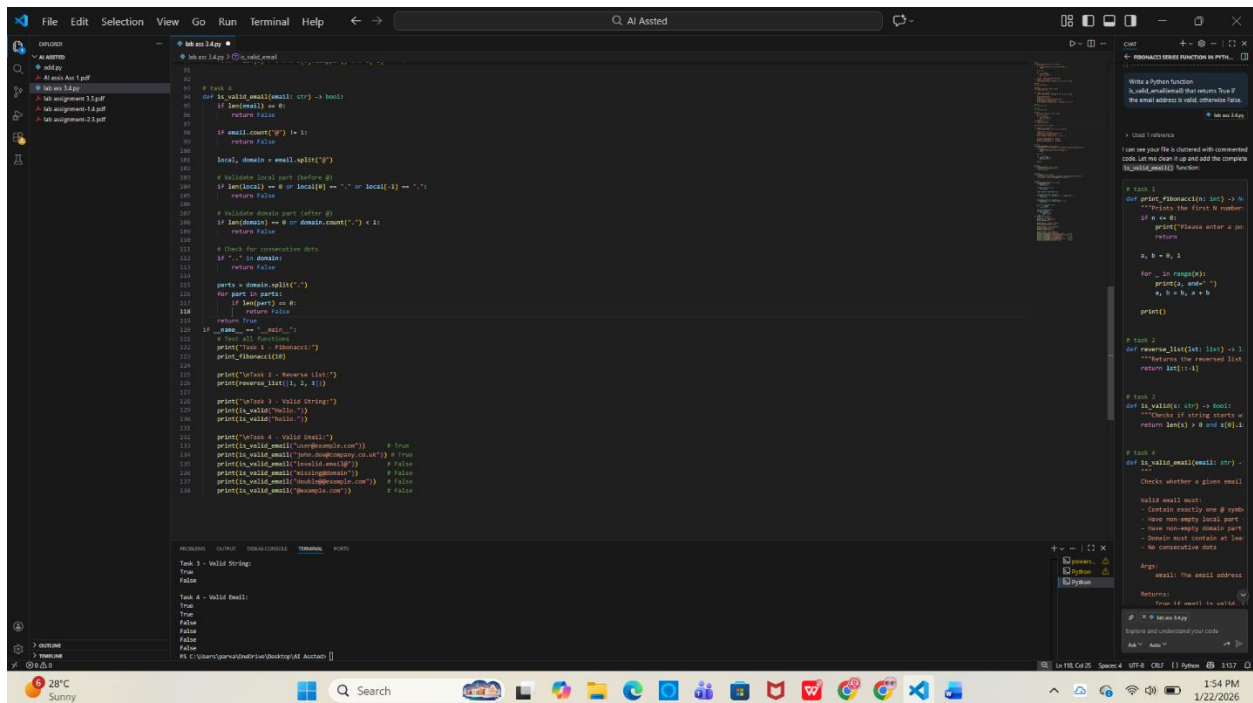
Expected Output #4

- Compare both outputs:

Zero-shot may result in basic or generic validation.

Few-shot gives detailed and specific logic (e.g., @ and domain checking).

- Submit both code versions and note how few-shot improves reliability.



The screenshot shows a VS Code editor with a Python script for email validation. The script defines a function `is_valid_email` that checks if a given email address is valid. It uses a few-shot prompt to guide the logic, including checks for local part, domain, and consecutive dots. The script also includes a `reverse_list` function and a `main` function that tests the `is_valid_email` function with various email addresses.

```
def is_valid_email(email: str) -> bool:
    """Returns True if the email address is valid, otherwise False"""

    # Task 1: Validate the email address
    if email.count("@") != 1:
        return False

    local, domain = email.split("@")

    # Task 2: Validate local part (before @)
    if len(local) == 0 or local[0] == "." or local[-1] == ".":
        return False

    # Task 3: Validate domain part (after @)
    if len(domain) == 0 or domain.count(".") < 1:
        return False

    # Task 4: Check for consecutive dots
    if "." in domain:
        return False

    parts = domain.split(".")
    for part in parts:
        if len(part) == 0:
            return False

    # Task 5: Check for valid domain
    if __name__ == "__main__":
        # Task 6: Test the function
        print("Task 1 - is_valid_email")
        print(reverse_list([1, 2, 3]))
        print(reverse_list([1, 2, 3]))
        print(reverse_list([1, 2, 3]))
        print("Task 2 - Valid String")
        print(is_valid_email("valid"))
        print(is_valid_email("invalid"))
        print(is_valid_email("user@example.com"))
        print(is_valid_email("john.doe@example.co.uk"))
        print(is_valid_email("invalid_email@"))
        print(is_valid_email("missingdomain"))
        print(is_valid_email("local@superdomain.com"))
        print(is_valid_email("example.com"))
```

The terminal output shows the results of the tests:

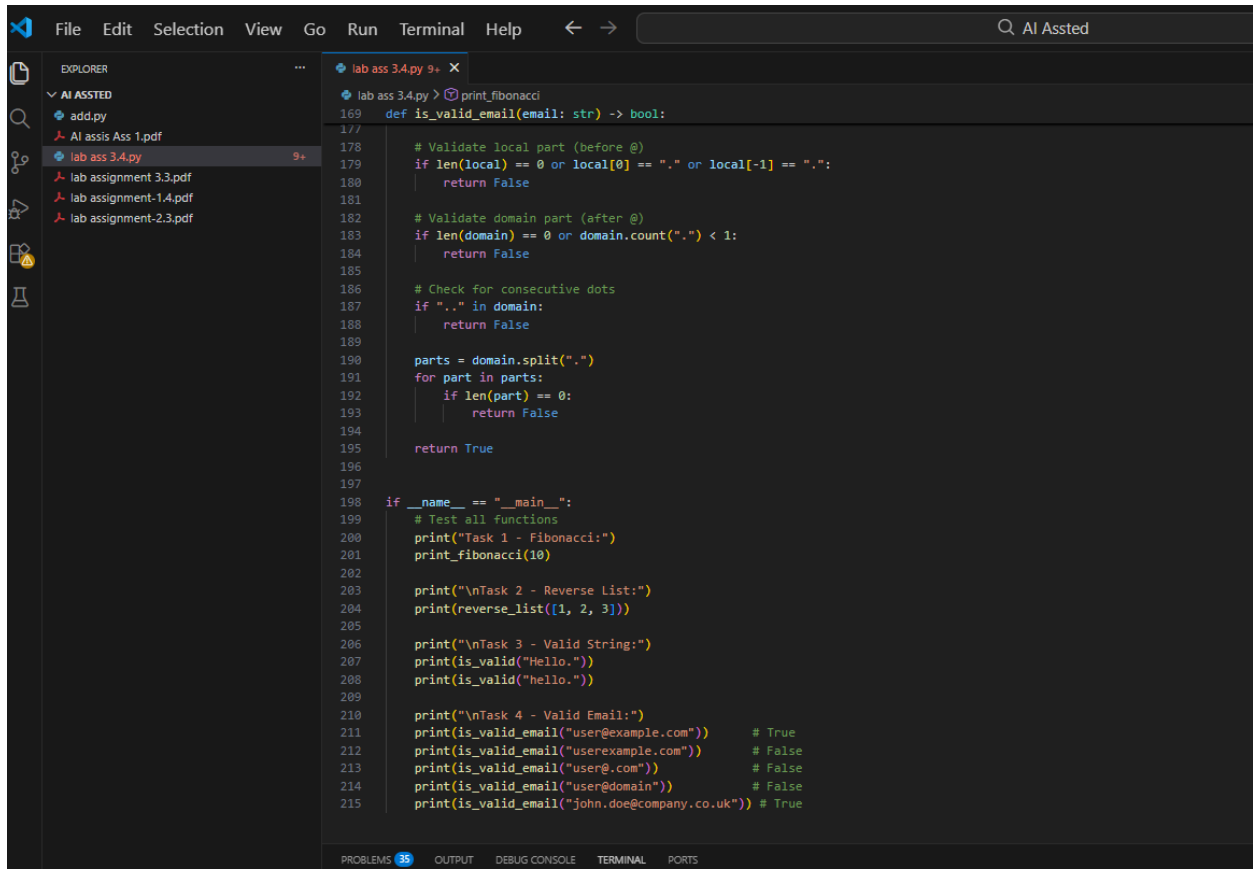
```
Task 1 - Valid String:
True
False

Task 2 - Valid Email:
True
False
False
False
True
False
False
False
True
```

```

139
140 #task4.2
141 # task 1
142 def print_fibonacci(n: int) -> None:
143     if n <= 0:
144         print("Please enter a positive integer.")
145         return
146
147     a, b = 0, 1
148
149     for _ in range(n):
150         print(a, end=" ")
151         a, b = b, a + b
152
153     print()
154
155
156 # task 2
157 def reverse_list(lst: list) -> list:
158     """Returns the reversed list."""
159     return lst[::-1]
160
161
162 # task 3
163 def is_valid(s: str) -> bool:
164     """Checks if string starts with capital letter and ends with period."""
165     return len(s) > 0 and s[0].isupper() and s[-1] == "."
166
167
168 # task 4
169 def is_valid_email(email: str) -> bool:
170     if len(email) == 0:
171         return False
172
173     if email.count("@") != 1:
174         return False
175
176     local, domain = email.split("@")

```



```
169 def is_valid_email(email: str) -> bool:
170     """
171     # Validate local part (before @)
172     if len(local) == 0 or local[0] == "." or local[-1] == ".":
173         return False
174
175     # Validate domain part (after @)
176     if len(domain) == 0 or domain.count(".") < 1:
177         return False
178
179     # Check for consecutive dots
180     if "." in domain:
181         return False
182
183     parts = domain.split(".")
184     for part in parts:
185         if len(part) == 0:
186             return False
187
188     return True
189
190
191 if __name__ == "__main__":
192     # Test all functions
193     print("Task 1 - Fibonacci:")
194     print_fibonacci(10)
195
196     print("\nTask 2 - Reverse List:")
197     print(reverse_list([1, 2, 3]))
198
199     print("\nTask 3 - Valid String:")
200     print(is_valid("Hello."))
201     print(is_valid("Hello."))
202
203     print("\nTask 4 - Valid Email:")
204     print(is_valid_email("user@example.com")) # True
205     print(is_valid_email("userexample.com")) # False
206     print(is_valid_email("user@.com")) # False
207     print(is_valid_email("user@domain")) # False
208     print(is_valid_email("john.doe@company.co.uk")) # True
```

Task 5: Prompt Tuning – Summing Digits of a Number

Task Description #5

- Experiment with 2 different prompt styles to generate a function that returns the sum of digits of a number.

Style 1: Generic task prompt

Style 2: Task + Input/Output example

Expected Output #5

- Two versions of the `sum_of_digits()` function.
- Example Output: `sum_of_digits(123) → 6`
- Short analysis: which prompt produced cleaner or more optimized code and why?

