

## Task 1: AI-Generated Logic for Reading Consumer Details

## Scenario

An electricity billing system must collect accurate consumer data.

The screenshot shows a code editor with two tabs: 'electricity\_billings.py' and 'electricity\_billings.ipynb'. The 'electricity\_billings.py' tab contains the following Python script:

```
electricity_billings.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(f"Type of Customer: {customer_type}")
19 print(f"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     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(f"Rate per unit: {rate_per_unit}")
32 print(f"Total Bill: {total_bill}")
```

The terminal below shows the script being run and outputting the results for a commercial customer.

```
PS C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding> cd "C:\Users\gbsgu\OneDrive\Documents\Desktop\Assisted coding"; C:\Users\gbsgu\AppData\Local\Programs\Python\Python311\python.exe electricity_billings.py
Enter previous units: 9
Enter current units: 9.8
Enter type of customer (residential/commercial): commercial
Previous Units: 9.0
Current Units: 9.8
Type of Customer: commercial
Units Consumed: 0.8
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.

The screenshot shows a Jupyter Notebook environment with several tabs open. The main code cell contains Python code for calculating electricity bills based on consumption levels and customer types. The code uses conditional statements to determine rates per unit and total bills. A sidebar panel titled 'Assisted coding' provides real-time feedback on the code's execution.

```
electricity_bill.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 # Read billing calculation (assuring rates)
22 if customer_type == "residential":
23     rate_per_unit = 5.0 + units_consumed
24     # Calculate energy charges for residential
25     # Using nested if statement for better readability
26     if units_consumed < 100:
27         energy_charges = units_consumed * 5.0
28     else:
29         energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
30
31     energy_charges = 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
32
33 elif customer_type == "commercial":
34     rate_per_unit = 10.0 + units_consumed
35
36     # Calculate energy charges for commercial
37     if units_consumed < 500:
38         energy_charges = 500 * 10.0 + (units_consumed - 500) * 12.0
39     else:
40         energy_charges = 500 * 10.0 + (units_consumed - 500) * 12.0 + (units_consumed - 500) * 15.0
41
42 elif customer_type == "industrial":
43     # Calculate energy charges with clear structure
44     if units_consumed < 500:
45         energy_charges = 500 * 10.0
46     else:
47         energy_charges = 500 * 10.0 + (units_consumed - 500) * 15.0
48
49 else:
50     print("Invalid customer type. Assuming residential rates")
51     rate_per_unit = 5.0
52
53 total_bill = units_consumed * rate_per_unit
54 print("Rate per unit: ", rate_per_unit)
55 print("Total Bill: ", total_bill)
56
57 print("Assuming residential rates")
58 if units_consumed < 100:
59     energy_charges = units_consumed * 5.0
60 else:
61     energy_charges = 100 * 5.0 + (units_consumed - 100) * 7.0
62
63 energy_charges = 100 * 5.0 + 200 * 7.0 + (units_consumed - 300) * 7.0
64
65 print("Energy Charges: ", energy_charges)
```

```

# electricity_billing.py
# Electricity Billing System
# This program calculates electricity bills using conditional statements for energy charges and fixed charges.

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

units_consumed = float(input("Enter previous units: "))
current_units = float(input("Enter current units: "))

energy_charges = 0.0
fixed_charge = 0.0

if customer_type == "domestic":
    if units_consumed <= 100:
        energy_charges = 100 * 3.0 + 5.0
    elif units_consumed > 100:
        energy_charges = 100 * 3.0 + 5.0 + (units_consumed - 100) * 5.0
    else:
        energy_charges = 100 * 3.0 + 5.0 + (units_consumed - 100) * 7.0
elif customer_type == "commercial":
    if units_consumed <= 200:
        energy_charges = 200 * 5.0 + 10.0
    elif units_consumed > 200:
        energy_charges = 200 * 5.0 + (units_consumed - 200) * 8.0
    else:
        energy_charges = 200 * 5.0 + (units_consumed - 200) * 12.0
else:
    print("Invalid customer type. Using domestic rates.")

print(f"Energy Charges (EC): ${energy_charges:.2f}")

```

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

### Scenario

Billing logic must be reusable for multiple consumers.

```

# 3.3.py
# Electricity Billing System with User-Defined Functions
# This program calculates electricity bills using modular functions for energy charges and Fixed charges

def calculate_energy_charges(units_consumed, customer_type):
    """
    Calculate energy charges based on customer type and units consumed.
    Uses tiered pricing structure for different consumption levels.
    """
    if customer_type == "domestic":
        # Domestic tiered pricing: 3.0 for first 100, 5.0 for next 200, 7.0 thereafter
        if units_consumed <= 100:
            return units_consumed * 3.0
        elif units_consumed <= 300:
            return 100 * 3.0 + (units_consumed - 100) * 5.0
        else:
            return 100 * 3.0 + 200 * 5.0 + (units_consumed - 300) * 7.0
    elif customer_type == "commercial":
        # Commercial tiered pricing: 5.0 for first 200, 8.0 for next 200, 12.0 thereafter
        if units_consumed <= 200:
            return units_consumed * 5.0
        elif units_consumed <= 400:
            return 200 * 5.0 + (units_consumed - 200) * 8.0
        else:
            return 200 * 5.0 + 200 * 8.0 + (units_consumed - 400) * 12.0
    else:
        print("Invalid customer type. Using domestic rates.")

def calculate_fixed_charge(customer_type):
    if customer_type == "commercial":
        return 10.0
    else:
        return 0.0

# Electricity Bill ...
Previous Units: 10.0
Units Consumed: 2.0
Customer Type: Commercial
Fixed Charges: $100.00
Total Bill: $116.00

print(calculate_energy_charges(100, "domestic"))
print(calculate_fixed_charge("commercial"))

```

```

3.3.py
4 def calculate_energy_charges(units_consumed, customer_type):
5     if units_consumed <= 300:
6         return 3.0 * units_consumed
7     elif units_consumed < 600:
8         return 180 + 3.0 * (units_consumed - 300)
9     else:
10        return 180 + 200 + 5.0 * (units_consumed - 600) + 7.0
11
12 def calculate_fixed_charges(customer_type):
13     Calculate fixed charges based on customer type.
14     Fixed charges are monthly service fees.
15
16     Args:
17         customer_type (str): Type of customer ('domestic', 'commercial', 'industrial')
18
19     Returns:
20         float: Fixed charges amount
21
22     if customer_type == "domestic":
23         return 50.0 # Monthly service fee for domestic customers
24     elif customer_type == "commercial":
25         return 180 # Monthly service fee for commercial customers
26     elif customer_type == "industrial":
27         return 200.0 # Monthly service fee for industrial customers
28     else:
29         # Default to domestic fixed charges
30         return 50.0
31
32 def main():
33     Main function to handle user input and display billing information.
34     calls the calculation functions and returns the calculated values.
35
36     # Read Previous Units (PU)
37     pu = float(input("Enter previous units: "))
38
39     # Read Current Units (CU)
40     cu = float(input("Enter current units: "))
41
42     # Read Type of Customer
43     customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()
44
45     # Calculate units consumed
46     units_consumed = cu - pu
47
48     # Calculate charges using functions
49     energy_charges = calculate_energy_charges(units_consumed, customer_type)
50     fixed_charges = calculate_fixed_charges(customer_type)
51
52     # Calculate total bill
53     total_bill = energy_charges + fixed_charges
54
55     # Display the results
56     print("Electricity Bill ---")
57     print(f"Previous Units: ({pu})")
58     print(f"Current Units: ({cu})")
59     print(f"Units Consumed: {units_consumed}")
60     print(f"Customer Type: {customer_type}({customer_type})")
61     print(f"Energy Charges: ${energy_charges:.2f}")
62     print(f"Fixed Charges: ${fixed_charges:.2f}")
63     print(f"Total Bill: ${total_bill:.2f}")
64
65     # Return calculated values as a tuple
66     return energy_charges, fixed_charges, total_bill
67
68 # Run the main function if this script is executed directly
69 if __name__ == "__main__":
70     main()

```

```

3.3.2.py
67 def main():
68     calls the calculation functions and returns the calculated values.
69
70     # Read Previous Units (PU)
71     pu = float(input("Enter previous units: "))
72
73     # Read Current Units (CU)
74     cu = float(input("Enter current units: "))
75
76     # Read Type of Customer
77     customer_type = input("Enter type of customer (domestic/commercial/industrial): ").lower()
78
79     # Calculate units consumed
80     units_consumed = cu - pu
81
82     # Calculate charges using functions
83     energy_charges = calculate_energy_charges(units_consumed, customer_type)
84     fixed_charges = calculate_fixed_charges(customer_type)
85
86     # Calculate total bill
87     total_bill = energy_charges + fixed_charges
88
89     # Display the results
90     print("Electricity Bill ---")
91     print(f"Previous Units: ({pu})")
92     print(f"Current Units: ({cu})")
93     print(f"Units Consumed: {units_consumed}")
94     print(f"Customer Type: {customer_type}({customer_type})")
95     print(f"Energy Charges: ${energy_charges:.2f}")
96     print(f"Fixed Charges: ${fixed_charges:.2f}")
97     print(f"Total Bill: ${total_bill:.2f}")
98
99     # Return calculated values as a tuple
100    return energy_charges, fixed_charges, total_bill
101
102 # Run the main function if this script is executed directly
103 if __name__ == "__main__":
104    main()

```

## Task 4: Calculation of Additional Charges

### Scenario

Electricity bills include multiple additional charges.

## Task 5: Final Bill Generation and Output Analysis

## Scenario

The final electricity bill must present all values clearly.

```

# Assisted coding
electricity_billing.py ④ 139

# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components
# Input validation and error handling for real-world applicability
try:
    # Read Previous Units (PU) with validation
    pu = float(input("Enter previous units: "))
    if pu < 0:
        raise ValueError("Previous units cannot be negative")
    # Read Current Units (CU) with validation
    cu = float(input("Enter current units: "))
    if cu < 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()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        exit(1)
    # Calculate electricity duty based on consumption
    units_consumed = cu - pu
    if units_consumed < 0:
        print("Error: Negative consumption (possible meter reset or error).")
        exit(1)
    if units_consumed < 10:
        # Residential: Current units are less than previous units.
        print("⚠️ A meter reading error has occurred. This may indicate a meter reading error or meter reset.")
        print("Setting energy charges to $0.00 for this billing cycle")
        electricity_duty = 0.0
        units_consumed_display = f"({units_consumed} U)"  # (adjusted to 0 for billing)
    else:
        # Industrial: Current units are more than previous units.
        electricity_duty = 0.6
        units_consumed_display = f"({units_consumed} U)"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Residential: EC = 0.08 * units_consumed
    # Commercial: EC = 0.10 * units_consumed
    # Industrial: EC = 0.10 * units_consumed
    energy_charge = 0.08 * units_consumed
    if customer_type == "commercial":
        energy_charge = 0.10 * units_consumed
    elif customer_type == "industrial":
        energy_charge = 0.10 * units_consumed
    # Calculate Electricity Duty (ED) as percentage of Energy Charge (EC)
    electricity_duty_rate = 0.05
    if customer_type == "domestic":
        electricity_duty_rate = 0.05
    elif customer_type == "commercial":
        electricity_duty_rate = 0.08
    elif customer_type == "industrial":
        electricity_duty_rate = 0.10
    # Calculate Total Bill (TB)
    total_bill = energy_charge + electricity_duty
    # Print Detailed Bill Summary
    print("Peter Reading Summary")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Customer Type: {customer_type}")
    print(f"Units Consumed: {units_consumed}")
    print("Electricity Bill Details")
    print(f"Energy Charge (EC): ${energy_charge:.2f}")
    print(f"Electricity Duty (ED): ${electricity_duty:.2f} (HRS)")
    print(f"Total Bill Amount: ${total_bill:.2f}")
    print("Bill Summary for Industrial Customer")
    print("Rate structure: Thread pricing applied")

```

```

# Assisted coding
electricity_billing.py ④ 139

# Electricity Billing System with Enhanced Accuracy and Real-world Features
# This program calculates comprehensive electricity bills with multiple charge components
# Input validation and error handling for real-world applicability
try:
    # Read Previous Units (PU) with validation
    pu = float(input("Enter previous units: "))
    if pu < 0:
        raise ValueError("Previous units cannot be negative")
    # Read Current Units (CU) with validation
    cu = float(input("Enter current units: "))
    if cu < 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()
    valid_types = ["domestic", "commercial", "industrial"]
    if customer_type not in valid_types:
        print(f"Invalid customer type '{customer_type}'. Valid options: {', '.join(valid_types)}")
        exit(1)
    # Calculate electricity duty based on consumption
    units_consumed = cu - pu
    if units_consumed < 0:
        print("Error: Negative consumption (possible meter reset or error).")
        exit(1)
    if units_consumed < 10:
        # Residential: Current units are less than previous units.
        print("⚠️ A meter reading error has occurred. This may indicate a meter reading error or meter reset.")
        print("Setting energy charges to $0.00 for this billing cycle")
        electricity_duty = 0.0
        units_consumed_display = f"({units_consumed} U)"  # (adjusted to 0 for billing)
    else:
        # Industrial: Current units are more than previous units.
        electricity_duty = 0.6
        units_consumed_display = f"({units_consumed} U)"
    # Calculate Energy Charges (EC) based on customer type and units consumed
    # Residential: EC = 0.08 * units_consumed
    # Commercial: EC = 0.10 * units_consumed
    # Industrial: EC = 0.10 * units_consumed
    energy_charge = 0.08 * units_consumed
    if customer_type == "commercial":
        energy_charge = 0.10 * units_consumed
    elif customer_type == "industrial":
        energy_charge = 0.10 * units_consumed
    # Calculate Electricity Duty (ED) as percentage of Energy Charge (EC)
    electricity_duty_rate = 0.05
    if customer_type == "domestic":
        electricity_duty_rate = 0.05
    elif customer_type == "commercial":
        electricity_duty_rate = 0.08
    elif customer_type == "industrial":
        electricity_duty_rate = 0.10
    # Calculate Total Bill (TB)
    total_bill = energy_charge + electricity_duty
    # Print Detailed Bill Summary
    print("Peter Reading Summary")
    print(f"Previous Units: {pu}")
    print(f"Current Units: {cu}")
    print(f"Customer Type: {customer_type}")
    print(f"Units Consumed: {units_consumed}")
    print("Electricity Bill Details")
    print(f"Energy Charge (EC): ${energy_charge:.2f}")
    print(f"Electricity Duty (ED): ${electricity_duty:.2f} (HRS)")
    print(f"Total Bill Amount: ${total_bill:.2f}")
    print("Bill Summary for Industrial Customer")
    print("Rate structure: Thread pricing applied")

```

The screenshot shows a Python code editor with two tabs: `electricity_bill.py` and `electricity_bill.py`. The code in `electricity_bill.py` is as follows:

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

```
# electric_bill.py X 1.0y
# electric_bill.py >
#     1 customer_type == 'domestic':
#     2     units_consumed = 10.0
#     3     customer_charge = 25.0
#     4     electricity_duty = 10.0
#     5     customer_charges = 40.0
#     6
#     7     units_consumed = 15.0
#     8     customer_charge = 35.0
#     9     customer_charge = 'commercial'
#    10     electricity_duty = 20.0
#    11     customer_charges = 40.0
#    12
#    13     units_consumed = 20.0
#    14     customer_charge = 50.0
#    15
#    16 # Calculate Total Bill Amount
#    17 total_bill = energy_charges + fixed_charges + customer_charges + electricity_duty
#    18
#    19 # Display the Input Summary
#    20 print("=> Meter Reading Summary ---")
#    21 print("Previous Units: (u.uF) ---")
#    22 print("Current Units: (u.uF) ---")
#    23 print("Customer Type: (customer_type.title())")
#    24 print("Units Consumed: (units_consumed.display())")
#    25
#    26 # Adding all charges for current consumption for billing accuracy
#    27 print("=> Electricity Bill Details ---")
#    28 print("Energy Charges (EC): (energy_charge)")
#    29 print("Fixed Charges (FC): (fixed_charge)")
#    30 print("Customer Charge (CC): (customer_charge)")
#    31 print("Electricity Duty (ED): (electricity_duty) (electricity_duty_rate*100.0f$B))")
#    32 print("Final Bill Amount: (total_bill)$B")
#    33
#    34 # Real world applicability notes
#    35 print("=> Bill Summary for (customer_type.title()) Customer ---")
#    36 print("Note: (customer_charge) is based on tiered pricing applied")
#    37 if units_consumed <= 0:
#    38     print("Error: (units_consumed) must be greater than zero")
#    39     print("Error: (units_consumed) must be less than or equal to 20.0")
#    40     print("Error: (units_consumed) must be less than or equal to 50.0")
#    41     print("Error: (units_consumed) must be less than or equal to 100.0")
#    42
#    43 # Additional billing accuracy checks
#    44 if units_consumed < 0:
#    45     print("Error: Current units are less than previous units. Please verify meter readings.")
#    46 if energy_charge < 0:
#    47     print("Error: Negative energy charges detected. Check input values.")
#    48
```

The terminal output shows the execution of the script and its results:

```
PROBLEMS OUTPUT DEBUGGING TERMINAL PIDS
```

```
PS C:\Users\ghaju\OneDrive\Documents\Desktop\Assisted coding> python C:/Users/ghaju/AppData/Local/Programs/Python/Python311/python.exe -c /Users/ghaju/OneDrive/Documents/Desktop/Assisted coding/electricity_bill.py
```

```
Enter previous units: 10
Enter current units: 20
Enter type of customer (domestic/commercial/industrial): industrial

--- Meter Reading Summary ---
Previous Units: 10.0
Current Units: 20.0
Customer Type: Industrial
Units Consumed: 10.0

--- Electricity Bill Details ---
Energy Charges (EC): $25.00
Fixed Charges (FC): $20.00
Customer Charge (CC): $40.00
Electricity Duty (ED): $10.00
Total Bill Amount: $75.00

--- Bill Summary for Industrial Customer ---
No 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  $N = 7 \rightarrow 0 1 1 2 3 5 8$
- Observation on how Copilot understood the instruction with zero context.

The screenshot shows the Visual Studio Code interface. In the center, there's a code editor with a Python file named 'lab ass 3.4.py'. The code defines a function 'print\_fibonacci' that prints the first N numbers of the Fibonacci series. A tooltip from GitHub Copilot provides a task description: "Write a Python function that takes an integer N as input and prints the first N numbers of the Fibonacci series starting from 0, following correct Fibonacci logic and clean coding practices." Below the code editor, the terminal shows the command 'PS C:\Users\parva\OneDrive\Desktop\AI Asssted>'. On the right side of the interface, there's a Chat window titled 'FIBONACCI SERIES FUNCTION IN PYTH...' which contains a message from Copilot: "I'll create a clean Python function that generates the first N numbers of the Fibonacci series:".

```
def print_fibonacci(n: int) -> None:
    if n <= 0:
        print("Please enter a positive integer.")
        return

    a, b = 0, 1

    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b

    print() # New line after the series
if __name__ == "__main__":
    print_fibonacci(10) # Output: 0 1 1 2 3 5 8 13 21 34
```

## 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

```

File Edit Selection View Go Run ...
EXPLORER
AI ASSTED
add.py
AI assis Ass 1.pdf
lab ass 3.4.py
lab assignment 3.3.pdf
lab assignment-1.4.pdf
lab assignment-2.3.pdf
...
lab ass 3.4.py > ...
# task 2
def reverse_list(lst: list) -> list:
    return lst[::-1]

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

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

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\parva\OneDrive\Desktop\AI Assted>

CHAT + ... | FIBONACCI SERIES FUNCTION IN PYTH...

Write a Python function that takes a list as input and returns the reversed list.  
Example:  
Input: [1, 2, 3]  
Output: [3, 2, 1]

I'll add a function to reverse a list to your existing file:

# ...existing code...

# task 2  
def reverse\_list(lst: list) -> list:  
 """  
 Takes a list as input and re...

Args:

Explore and understand your code

Ask Auto

Ln 23, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.13.7 1:40 PM 1/22/2026

### 1 and 2 tasks output

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\parva\OneDrive\Desktop\AI Assted> & C:/Users/parva/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/parva/OneDrive/Desktop/AI Assted/lab ass 3.4.py"

0 1 1 2 3 5 8 13 21 34  
0 1 1 2 3 5 8 13 21 34  
Input: [1, 2, 3]  
Output: [3, 2, 1]

PS C:\Users\parva\OneDrive\Desktop\AI Assted>

## 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.

```
File Edit Selection View Go Run Terminal Help < > Q: AI Assisted D X ... CHAT + x ⌂ ... FIBONACCI SERIES FUNCTION IN PYTH... task 3 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 capital letter and ends with a period. len(s) > 0 and s[0].isupper() and s[-1] == '.' _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
```

NameError: name 'print\_fibonacci' is not defined  
PS C:\Users\parva\OneDrive\Desktop\AI Assisted & c:\Users\parva\AppData\Local\Programs\Python\Python313\python.exe c:/Users/parva/OneDrive/Desktop/AI Assisted/Lab\_ass\_3.4.py  
0 1 1 2 3 5 8 13 21 34  
Input: [1, 2, 3]  
Output: [3, 2, 1]  
True  
False  
False  
PS C:\Users\parva\OneDrive\Desktop\AI Assisted>

POWERShell Python Python

Explore and understand your code Ask Auto

28°C Sunny

In 51, Col 1 Spaces: 4 UTF-8 CRLF () Python 3.13.7 1:48 PM 1/22/2026

## 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.

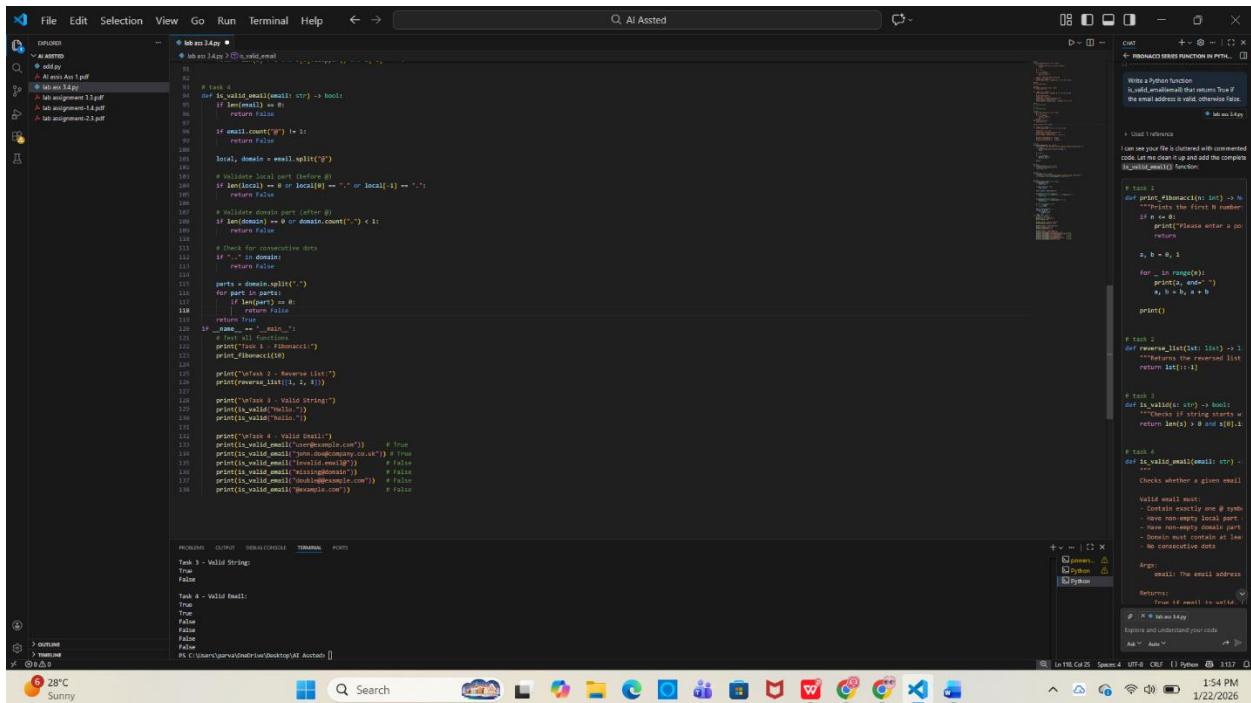
## Expected Output #4

- 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.



```
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 "@"
```

The screenshot shows a code editor interface with a dark theme. On the left is the Explorer sidebar, which lists files like 'AI ASSTED', 'add.py', 'AI assis 1.pdf', 'lab assignment 3.3.pdf', 'lab assignment-1.4.pdf', and 'lab assignment-2.3.pdf'. The main area displays a Python script named 'lab ass 3.4.py' with the following content:

```
def is_valid_email(email: str) -> bool:
    """
    Validate email address
    """
    if len(local) == 0 or local[0] == "." or local[-1] == ".":
        return False
    if len(domain) == 0 or domain.count(".") < 1:
        return False
    if ".." in domain:
        return False
    parts = domain.split(".")
    for part in parts:
        if len(part) == 0:
            return False
    return True

if __name__ == "__main__":
    # Test all functions
    print("Task 1 - Fibonacci:")
    print_fibonacci(10)

    print("\nTask 2 - Reverse List:")
    print(reverse_list([1, 2, 3]))

    print("\nTask 3 - Valid String:")
    print(is_valid("Hello."))
    print(is_valid("hello."))

    print("\nTask 4 - Valid Email:")
    print(is_valid_email("user@example.com"))      # True
    print(is_valid_email("user@example.com"))        # False
    print(is_valid_email("user@.com"))               # False
    print(is_valid_email("user@domain"))             # False
    print(is_valid_email("john.doe@company.co.uk")) # True
```

At the bottom, there are tabs for PROBLEMS (35), OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS.

## 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?

Screenshot of VS Code showing AI Assisted code completion for `lab ass 3.4.py`. The code implements several tasks:

- `#task 1`: `def print_fibonacci(n: int) -> None:`
- `#task 2`: `def reverse_list(lst: list) -> list:`
- `#task 3`: `def is_valid(s: str) -> bool:`
- `#task 4`: `def sum_of_digits(n: int) -> int:`

The terminal shows the command `PS C:\Users\parva\OneDrive\Desktop\AI Assisted> & C:\users\parva\appdata\local\Programs\Python\Python313\python.exe "c:/users/parva/OneDrive/Desktop/AI Assisted/lab ass 3.4.py"` and output indicating Task 1 - Fibonaci completed.

Screenshot of VS Code showing AI Assisted code completion for `lab ass 3.4.py`. The code now includes all five tasks from the previous screenshot plus additional ones:

- `#task 1`: `def print_fibonacci(n: int) -> None`
- `#task 2`: `def reverse_list(lst: list) -> list`
- `#task 3`: `def is_valid(s: str) -> bool`
- `#task 4`: `def sum_of_digits(n: int) -> int`
- `#task 5`: `def is_valid_email(email: str) -> bool`

The terminal shows the command `PS C:\Users\Varsha\OneDrive\Desktop\AI Assisted> & C:\users\Varsha\appdata\local\Programs\Python\Python313\python.exe "c:/users/Varsha/OneDrive/Desktop/AI Assisted/lab ass 3.4.py"` and output indicating Task 1 - Fibonaci completed.