

2303A1739

Gangula Navyasri

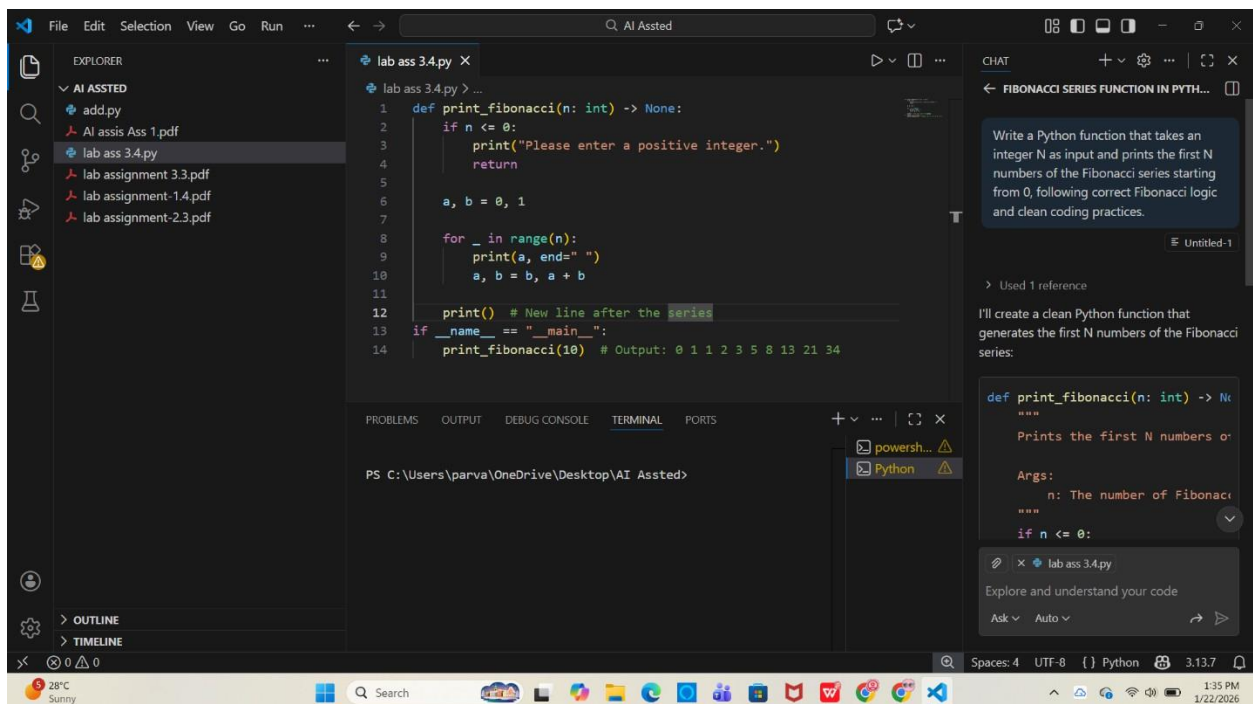
## Task 1: Zero-shot Prompt – Fibonacci Series Generator

### 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

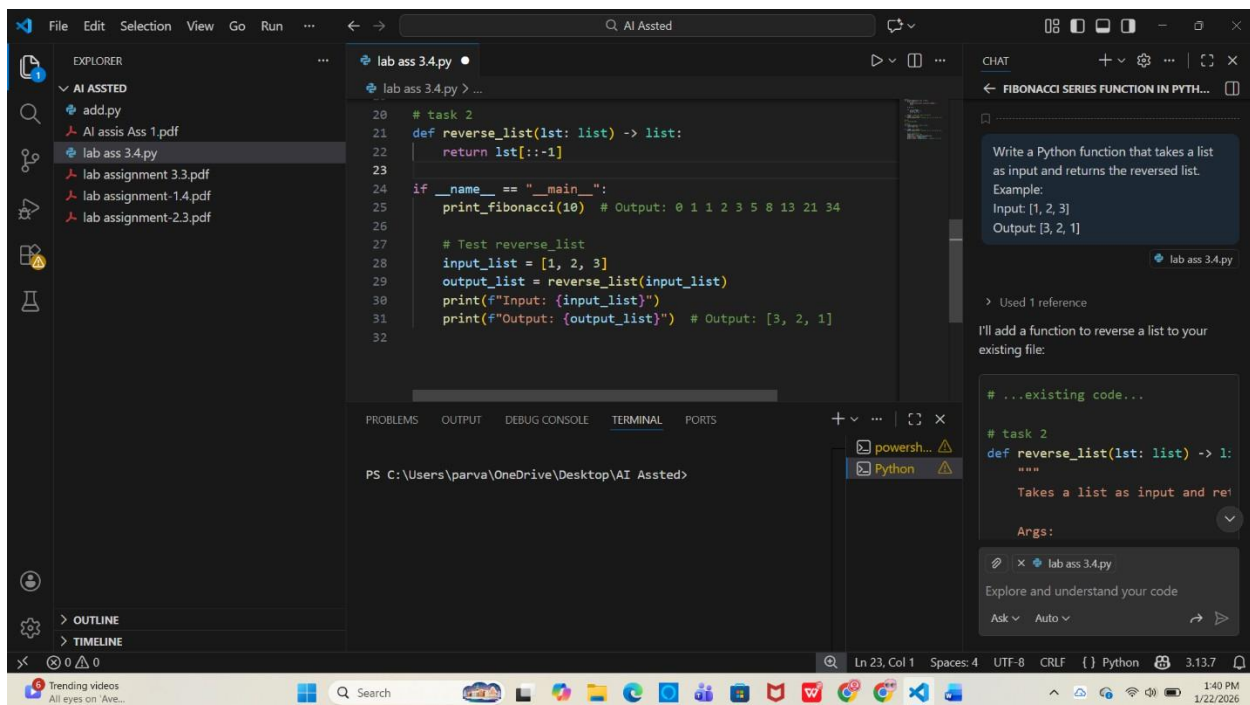
### Task Description #2

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

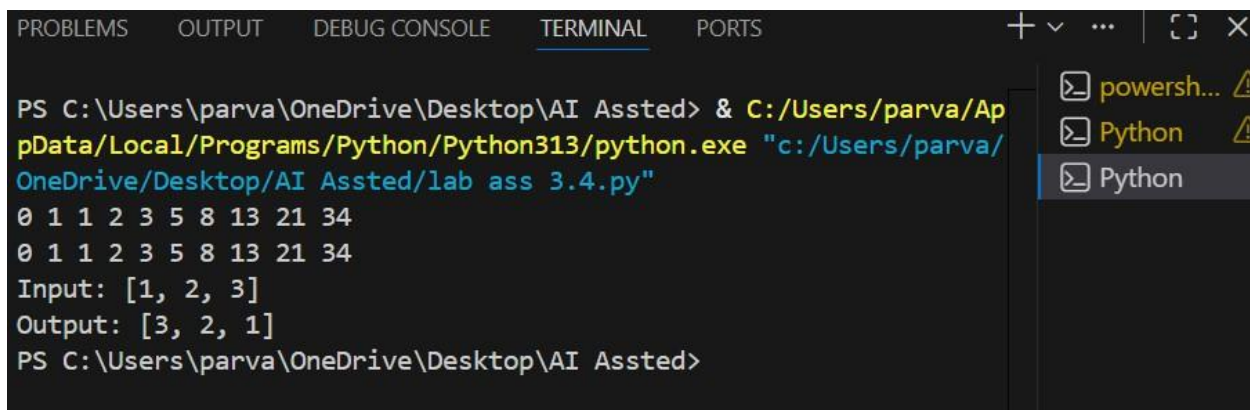
Expected Output #2

- 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



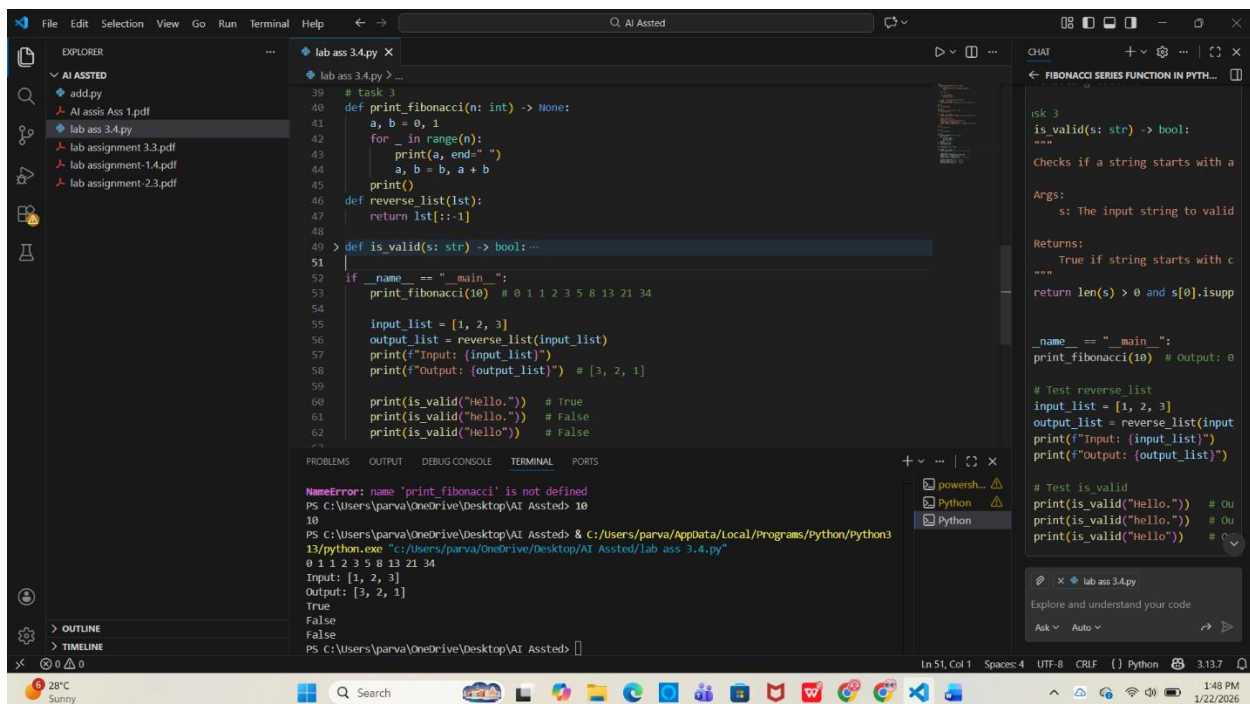
## Task 3: Few-shot Prompt – String Pattern Matching

### 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(list):
    return list[::-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 and ends with .
    """
    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.

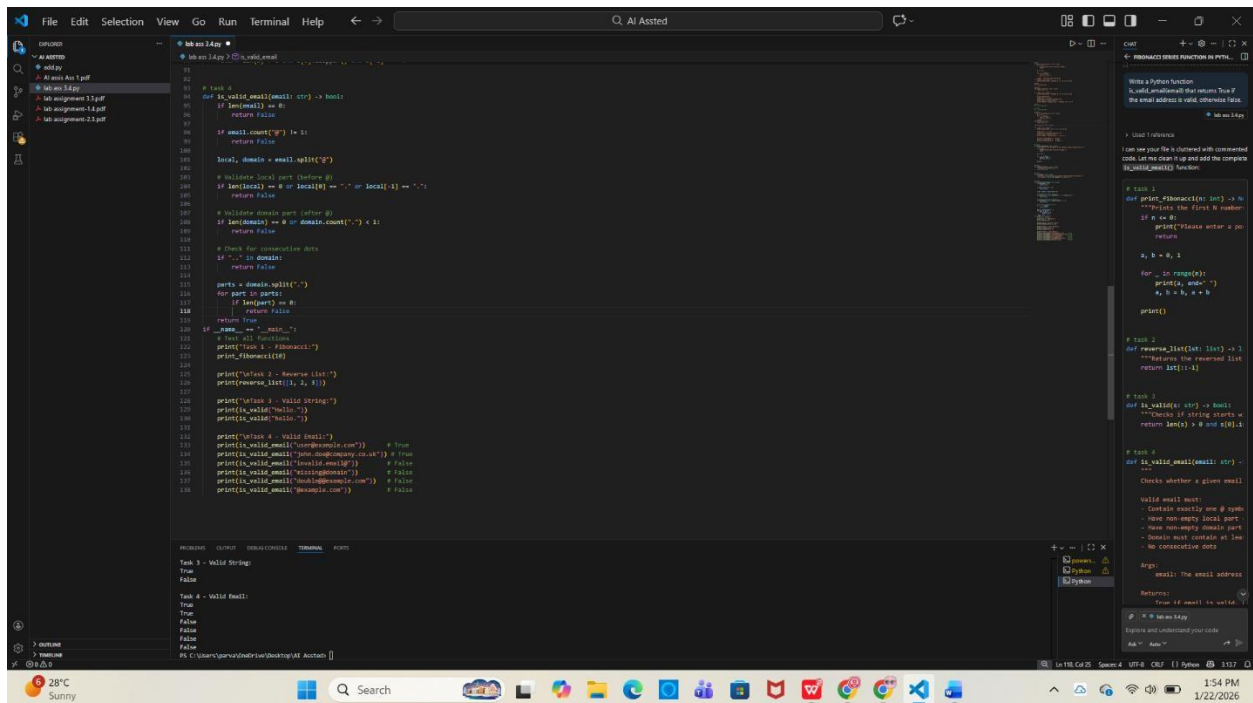
## Expected Output #4

- 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 an email is valid based on several criteria: it must contain exactly one '@' symbol, have a non-empty local part, a non-empty domain part, and a domain that contains at least one dot. 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."""
    # Check for exactly one '@'
    if email.count("@") != 1:
        return False

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

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

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

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

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

    return True

# Test all functions
print("Task 1 - Reverse List:")
print(reverse_list([1, 2, 3]))

print("Task 2 - Valid String:")
print(is_valid("hello"))
print(is_valid("hello "))

print("Task 3 - Valid Email:")
print(is_valid_email("user@example.com")) # True
print(is_valid_email("john.doe@company.co.uk")) # True
print(is_valid_email("invalid_email")) # False
print(is_valid_email("missingdomain")) # False
print(is_valid_email("malformed@domain.com")) # False
print(is_valid_email("example.com")) # False
```

The terminal output shows the results of the tests:

```
Task 1 - Reverse List:
True
False

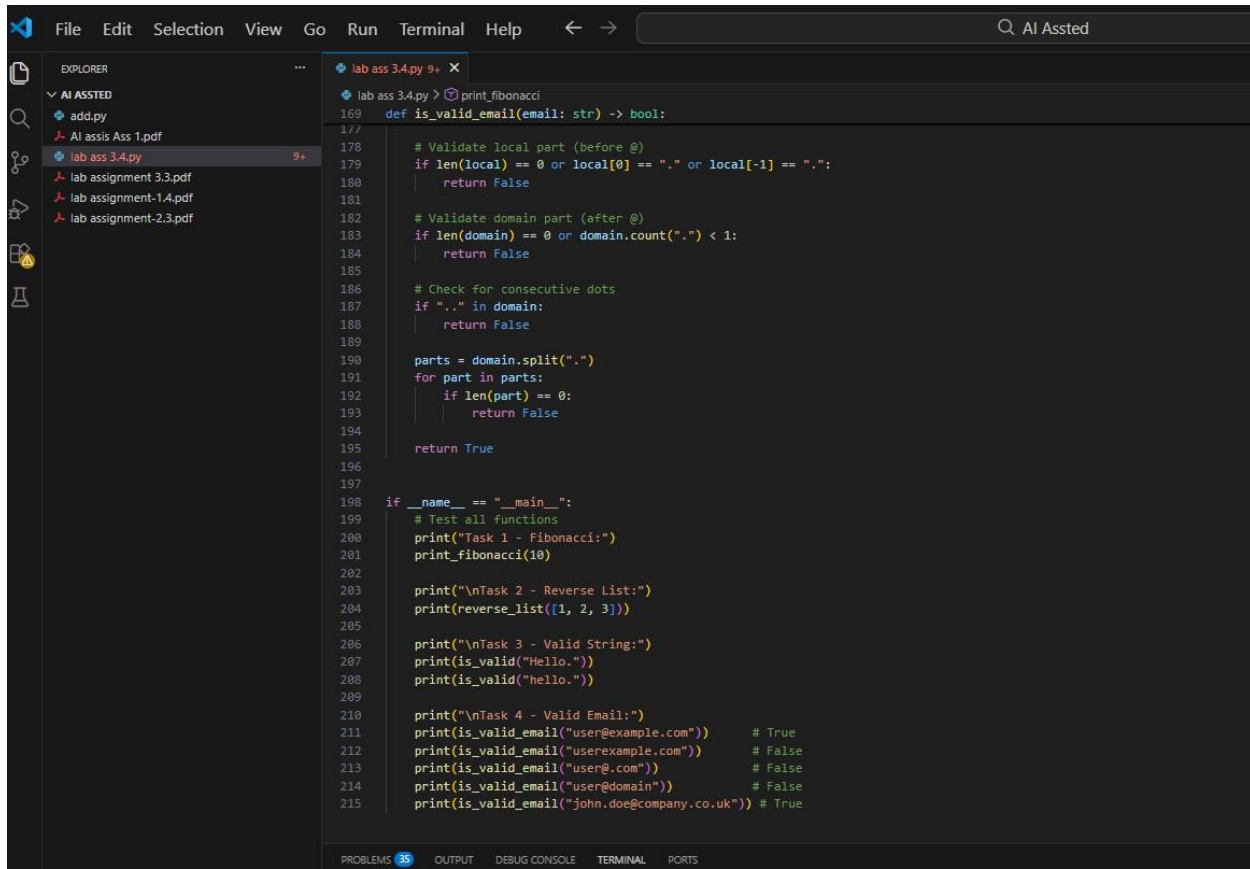
Task 2 - Valid String:
True
False

Task 3 - Valid Email:
True
False
False
False
False
```

```

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?



