

Name : R. Shashideepika

H NO:2303A51871

Batch-13

ASSIGNMENT-3.4

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.

PROMPT:

```
#Generate a Python function that takes an integer N as input and prints the first N Fibonacci numbers in order.
```

CODE:

```
#Generate a Python function that takes an integer N as input and prints the first N Fibonacci numbers in order.
def print_fibonacci(N):
    a, b = 0, 1
    for _ in range(N):
        print(a, end=' ')
        a, b = b, a + b
    print() # for a new line after printing all Fibonacci numbers
# Example usage:
print_fibonacci(10) # This will print the first 10 Fibonacci numbers
```

OUTPUT:

```
-3.4.py
0 1 1 2 3 5 8 13 21 34
PS C:\Users\shash\AppData\Local\Programs\Microsoft VS Code> █
```

JUSTIFICATION:

The function initializes the first two Fibonacci numbers (0 and 1) and uses a loop that runs N times to generate the sequence. In each iteration, the current number is printed and the next Fibonacci values are calculated using efficient variable updates. This iterative approach avoids extra memory usage and correctly prints the first N Fibonacci numbers in order.

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.

Prompt:

```
# Write a Python function that reverses a given list.
```

```
# Example: input [1, 2, 3] → output [3, 2, 1]
```

Code:

```
# Write a Python function that reverses a given list.
# Example: input [1, 2, 3] → output [3, 2, 1]

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

# Example usage:
print(reverse_list([1, 2, 3]))  # This will print [3, 2, 1]
```

Output:

```
[3, 2, 1]
PS C:\Users\shash\AppData\Local\Programs\Microsoft VS Code> █
```

Justification:

The function `reverse_list(lst)` reverses a list using Python's slicing technique `[::-1]`. This method creates a new list by traversing the original list from the last element to the first. It is concise, readable, and efficient, making it a reliable way to reverse a list without using loops or extra logic.

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.

Prompt:

```
# Write a Python function is_valid(text) that returns True if a string
# starts with a capital letter and ends with a period, otherwise False.

# Example: "Hello." → True
# Example: "hello." → False
# Example: "Hello" → False
```

CODE:

```
# Write a Python function is_valid(text) that returns True if a string
# starts with a capital letter and ends with a period, otherwise False.
# Example: "Hello." → True
# Example: "hello." → False
# Example: "Hello" → False

def is_valid(text):
    if not text:
        return False
    return text[0].isupper() and text[-1] == '.'
print(is_valid("Hello."))
print(is_valid("hello."))
print(is_valid("Hello"))
```

OUTPUT:

```
-3.4.py
True
False
False
```

JUSTIFICATION:

The function first checks if the input string is empty to avoid index errors. It then verifies two conditions: the first character must be an uppercase letter and the last character must be a period (.). By using built-in string methods, the function efficiently determines whether the given text meets the required format and returns the correct Boolean result.

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.

PROMPT:

```
# Write a Python function that checks whether an email address is valid and returns True or False.
```

CODE:

```
Write a Python function that checks whether an email address is valid and returns True or False.
# Write a Python function that checks whether an email address is valid and returns True or False.
def is_valid_email(email):
    if "@" not in email:
        return False
    username, domain = email.split("@", 1)
    if username == "":
        return False
    if "." not in domain or domain.startswith("."):
        return False
    return True
email = input("Enter email address: ")
print(is_valid_email(email))
```

OUTPUT:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter email address: reddy@gmail.com
True
PS C:\Users\neera\OneDrive\Desktop\assignments>
```

PROMPT:

```
# Write a Python function that checks whether an email address is valid and returns True or False.
```

```
# Example: "user@gmail.com" → True
```

```
# Example: "user@gmai" → False
```

```
# Example: "user@.com" → False
```

CODE:

```
# Write a Python function that checks whether an email address is valid and returns True or False.  
# Example: "user@gmail.com" → True  
# Example: "user@gmail" → False  
# Example: "user@.com" → False  
def is_valid_email(email):  
    if "@" not in email:  
        return False  
    username, domain = email.split("@", 1)  
    if username == "":  
        return False  
    if "." not in domain or domain.startswith("."):   
        return False  
    return True  
  
# Example usages:  
print(is_valid_email("user@gmail.com"))  
print(is_valid_email("user@gmail"))  
print(is_valid_email("user@.com"))
```



OUTPUT:

```
-3.4.py  
True  
False  
False
```

JUSTIFICATION:

Zero-shot prompting provides only the task description, so Copilot generates a simple and generic email validation logic. This approach may miss several invalid cases. In few-shot prompting, example inputs and outputs are given along with the task. These examples help Copilot understand the validation rules more clearly. As a result, few-shot prompting produces more accurate and reliable code than zero-shot prompting.

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?

PROMPT:

#Write a Python function that returns the sum of digits of a number.

CODE:

```
#Write a Python function that returns the sum of digits of a number.
def sum_of_digits(n):
    if n<0:
        n = -n
    total = 0
    while n > 0:
        digit = n % 10
        total += digit
        n //= 10
    return total
n=int(input("Enter a number: "))
print(sum_of_digits(n))
```

OUTPUT:

```
-3.4.py
Enter a number: 123
6
```

Style 2: Task + Input/Output example

PROMPT

```
# Write a Python function that returns the sum of digits of a number.

# Example: sum_of_digits(123) → 6
```

CODE:

```
# Write a Python function that returns the sum of digits of a number.
# Example: sum_of_digits(123) → 6

def sum_of_digits(n):
    if n < 0:
        n = -n
    total = 0
    while n > 0:
        digit = n % 10
        total += digit
        n //= 10
    return total
print(sum_of_digits(123))
print(sum_of_digits(456))
print(sum_of_digits(789))
```

OUTPUT:

```
-3.4.py
6
15
24
```

JUSTIFICATION:

The generic prompt produced a correct but longer solution using a loop.

The prompt with an example guided Copilot to generate a cleaner and more optimized one-line solution.

Providing input/output examples helps the model infer intent more clearly and produce concise, efficient code.