ASSIGNMENT-03

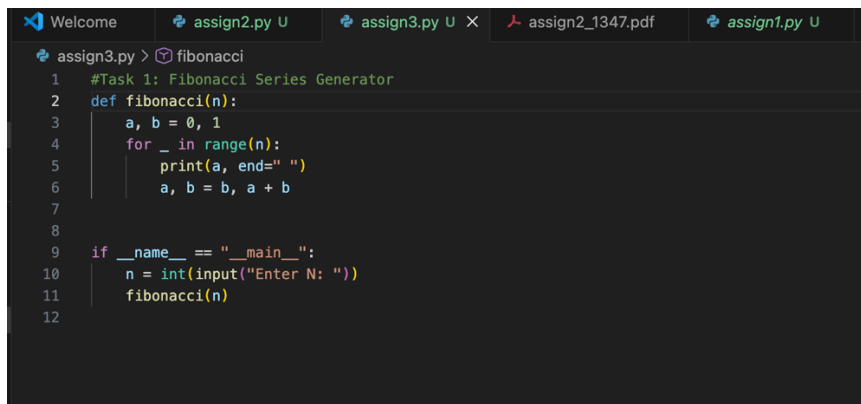**Name:** Chitti Udaya Sathwika

HT. No: 2303A51322

Batch: 19

Task 1: Zero-shot Prompt – Fibonacci Series Generator

Scenario : In this task, a zero-shot prompting technique was used. A single comment prompt was written without providing any examples, instructing GitHub Copilot to generate a Python function that prints the first N Fibonacci numbers.

Prompt: # Write a Python function to print the first N Fibonacci numbers

Code:

```python
#Task 1: Fibonacci Series Generator
def fibonacci(n):
    a, b = 0, 1
    for _ in range(n):
        print(a, end=" ")
        a, b = b, a + b


if __name__ == "__main__":
    n = int(input("Enter N: "))
    fibonacci(n)
```

Result:

```
Enter N: 7
0 1 1 2 3 5 8
```

Observation:

The zero-shot prompt was sufficient for Copilot to correctly infer the Fibonacci logic, even without any examples or additional context. However, the function behavior depended heavily on Copilot's prior training, and the output format was assumed rather than explicitly defined. This shows that zero-shot prompting works well for well-known problems but may lack consistency for ambiguous or complex tasks.

Task 2: One-shot Prompt – List Reversal Function. In this task, a one-shot prompting approach was used by providing a single example along with the instruction to help Copilot generate a correct list reversal function

Prompt: # Write a Python function to reverse a list

Example: input [1, 2, 3] -> output [3, 2, 1]

Code:

```
#Task 2: List Reversal Function
def reverse_list(lst):
    return lst[::-1]


if __name__ == "__main__":
    lst = list(map(int, input("Enter list elements: ").split()))
    print(reverse_list(lst))
```

Result:

```
Enter N: 7
0 1 1 2 3 5 8 Enter list elements: 1 2 3
[3, 2, 1]
```

Observation:
Providing one example significantly improved Copilot's accuracy and confidence in choosing an optimal approach. The generated solution was concise and efficient, using Python slicing. Compared to zero-shot prompting, one-shot prompting reduced ambiguity and guided Copilot toward the expected output format and logic

Task 3: Few-shot Prompt – String Pattern Matching

Scenario: This task used a few-shot prompting technique by providing multiple examples to help Copilot understand a specific string validation pattern.

Prompt: # Write a function to check if a string starts with a capital letter and ends with a period

Example: "Hello." -> True

Example: "hello." -> False

Example: "Hello" -> False

Code:

```python
#Task 3: String Pattern Matching
def is_valid(s):
    if len(s) == 0:
        return False
    return s[0].isupper() and s.endswith(".")


if __name__ == "__main__":
    s = input("Enter string: ")
    print(is_valid(s))
```

Result:

```
Enter N: 7
0 1 1 2 3 5 8 Enter list elements: 1 2 3
[3, 2, 1]
Enter string: Hello.
True
```

Observation:
The few-shot prompt enabled Copilot to accurately identify the string pattern requirements and generate a precise validation function. The multiple examples clarified edge cases and reduced misinterpretation. This demonstrates that few-shot prompting is highly effective when pattern recognition or conditional logic is involved.

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

You are participating in a code review session. This task compares zero-shot and few-shot prompting by generating two versions of an email validation function and analyzing their differences

Prompt: Zero-Shot Prompt: # Write a Python function to validate an email address

Prompt: Few-Shot Prompt: # Write a Python function to validate an email address

 # Example: "test@gmail.com" -> True

 # Example: "testgmail.com" -> False

 # Example: "test@com" -> False

Code:

```python
#Task 4: Email Validator
#Zero-shot Version
def validate_email(email):
    return "@" in email


if __name__ == "__main__":
    email = input("Enter email: ")
    print(validate_email(email))

#Few-shot Version
def validate_email(email):
    if "@" not in email:
        return False

    username, domain = email.split("@", 1)

    if username == "":
        return False

    if "." not in domain:
        return False

    return True


if __name__ == "__main__":
    email = input("Enter email: ")
    print(validate_email(email))
```

Result:

```
Enter N: 7
0 1 1 2 3 5 8 Enter list elements: 1 2 3
[3, 2, 1]
Enter string: Hello.
True
Enter email: test@gmail.com
True
Enter email: predict@gmail.com
True
```

Observation

The zero-shot version produced a very basic and unreliable validation logic, while the few-shot prompt resulted in a more structured and realistic solution. The examples guided Copilot to include domain checks and input validation, significantly improving reliability. This comparison clearly highlights the advantage of few-shot prompting for real-world validation tasks.

Task 5: Prompt Tuning – Summing Digits of a Number. In this task, two different prompt styles were used to study how prompt tuning affects code quality and optimization.

Prompt: Style-1:Generic Prompt # Write a function to return the sum of digits of a number

Prompt with I/O Example: # Write a function to return the sum of digits of a number

 # Example: sum_of_digits(123) -> 6

Code:

```
#Task 5: Sum of Digits
#Style-1
def sum_of_digits(n):
    total = 0
    for digit in str(n):
        total += int(digit)
    return total


if __name__ == "__main__":
    n = int(input("Enter number: "))
    print(sum_of_digits(n))


#Style-2
def sum_of_digits(n):
    return sum(int(digit) for digit in str(n))


if __name__ == "__main__":
    n = int(input("Enter number: "))
    print(sum_of_digits(n))
```

Result:

```
Enter N: 7
0 1 1 2 3 5 8 Enter list elements: 1 2 3
[3, 2, 1]
Enter string: Hello.
True
Enter email: test@gmail.com
True
Enter email: predict@gmail.com
True
Enter number: 123
6
Enter number: 143
8
```

Observation:

The prompt that included an input-output example produced a cleaner and more optimized implementation. The example encouraged Copilot to generate concise and Pythonic code using built-in functions. This demonstrates that prompt tuning with examples not only improves correctness but also enhances code quality and efficiency.