

AI ASSISTED CODING

ASSIGNMENT-9.2

B.Sri Laxmi Gayathri

2303a52033

Batch:38

Lab 9 – Documentation Generation: Automatic Documentation and Code Comments

Lab Objectives:

- To use AI-assisted coding tools for generating Python documentation and code comments.
- To apply zero-shot, few-shot, and context-based prompt engineering for documentation creation.
- To practice generating and refining docstrings, inline comments , and module-level documentation.
- To compare outputs from different prompting styles for quality analysis.

Task Description -1(Documentation – Function Summary Generation)

Task:

Use AI to generate concise functional summaries for each Python function in a given script.

Instructions:

- Provide a Python script to the AI.
- Ask the AI to write a short summary describing the purpose of each function.
- Ensure summaries are brief and technically accurate.
- Do not include code implementation details.

Expected Output -1:

A Python script where each function contains a clear and concise summary explaining its purpose.

Prompt:

Write a robust Python function named `generate_summary(documentation, max_sentences=3)` that returns the first `max_sentences` sentence-like segments from a documentation string.

Code & Output:

```
#documentation function summary generation
def generate_summary(documentation):
    # Split the documentation into sentences
    sentences = documentation.split('. ')

    # Select the first few sentences as the summary (you can adjust this as needed)
    summary = '. '.join(sentences[:3]) # Adjust the number of sentences as needed

    return summary

# Example usage
documentation = """This function takes a list of numbers and returns the average. It first checks if the list is empty to avoid division by zero.
summary = generate_summary(documentation)
print("Summary:", summary)
```

```
Summary: This function takes a list of numbers and returns the average. It first checks if the list is empty to avoid division by zero.
```

Explanation:

This task demonstrates how AI can assist in improving code documentation by automatically generating concise functional summaries. Instead of manually writing documentation, the AI analyzes each function and produces short, technically accurate descriptions of its purpose. The summaries focus only on what the function does, without including implementation details.

Task Description -2 (Documentation – Logical Explanation for Conditions and Loops)

Task:

Use AI to document the logic behind conditional statements and loops in a Python program.

Instructions:

- Provide a Python program without comments.
- Instruct AI to explain only decision-making logic and loop behavior.

- Skip basic syntax explanations.

Expected Output -2:

Python code with clear explanations describing the logic of conditions and loops.

Prompt:

Add clear and concise explanations describing only the logic behind conditional statements and loops. Focus on decision-making and iteration behavior. Do not explain basic syntax or rewrite the code.

Code & Output:

```
#conditions and loops
# if statement
x = 10
if x > 5:
    print("x is greater than 5")
# if-else statement
y = 3
if y % 2 == 0:
    print("y is even")
else:
    print("y is odd")
# if-elif-else statement
z = 15
if z < 10:
    print("z is less than 10")
elif z < 20:
    print("z is between 10 and 20")
else:
    print("z is 20 or greater")
# for loop
for i in range(5):
    print("Iteration:", i)
# while loop
count = 0
while count < 5:
    print("Count:", count)
    count += 1
```

```
x is greater than 5
y is odd
z is between 10 and 20
Iteration: 0
Iteration: 1
Iteration: 2
Iteration: 3
Iteration: 4
Count: 0
Count: 1
Count: 2
Count: 3
Count: 4
```

Explanation:

This task demonstrates how AI can improve code understanding by documenting the reasoning behind conditions and loops. Instead of explaining syntax, the AI focuses on why decisions are made and how loops control program flow. This enhances clarity and logical comprehension of the program.

Task Description -3 (Documentation – File-Level Overview)**Task:**

Use AI to generate a high-level overview describing the functionality of an entire Python file.

Instructions:

- Provide the complete Python file to AI.
- Ask AI to write a brief overview summarizing the file's purpose and functionality.
- Place the overview at the top of the file.

Expected Output -3:

A Python file with a clear and concise file-level overview at the beginning.

Prompt:

To generate a clear, concise, and technically accurate file-level overview that summarizes the overall purpose and functionality of the program.

Code & Output:

```
#level overview
import pygame
class LevelOverview:
    def __init__(self, screen, levels):
        self.screen = screen
        self.levels = levels
        self.font = pygame.font.SysFont(None, 36)
        self.selected_level = 0

    def draw(self):
        self.screen.fill((255, 255, 255))
        for i, level in enumerate(self.levels):
            color = (0, 255, 0) if i == self.selected_level else (0, 0, 0)
            text = self.font.render(f"Level {i + 1}", True, color)
            self.screen.blit(text, (50, 50 + i * 40))

    def handle_input(self, event):
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_UP:
                self.selected_level = max(0, self.selected_level - 1)
            elif event.key == pygame.K_DOWN:
                self.selected_level = min(len(self.levels) - 1, self.selected_level + 1)
            elif event.key == pygame.K_RETURN:
                return self.levels[self.selected_level]
        return None
```

```
pygame 2.6.1 (SDL 2.28.4, Python 3.12.12)
Hello from the pygame community. https://www.pygame.org/contribute.html
```

COMPILEWINDOWS — WIKI

COMPILE PYGAME ON WINDOWS

There are four steps.

- Get a C/C++ compiler.
- Install Python 3.6+
- Checkout pygame from github.
- Run the pygame install commands.

Explanation:

This task demonstrates how AI can assist in generating high-level documentation for an entire Python file. Instead of manually summarizing the program, the AI analyzes the overall structure and functionality to produce a concise overview. The file-level documentation helps readers quickly understand the program's purpose and key features without reviewing the full implementation.

Task Description -4 (Documentation – Refine Existing Documentation)

Task:

Use AI to improve clarity and consistency of existing documentation in Python code.

Instructions:

- Provide Python code containing basic or unclear comments.
- Ask AI to rewrite the documentation to improve clarity and consistency.
- Ensure technical meaning remains unchanged.

Expected Output -4:

Python code with refined and improved documentation that is clear and consistent.

Prompt:

A Python script that contains basic, unclear, or inconsistent comments and documentation. Your task is to rewrite and refine the existing documentation to improve clarity, readability, and consistency.

Code & Output:

```
def calculate_discount(price, rate):
    """
    Calculates the final price after applying a discount rate
    to the original price.
    """
    discount = price * rate
    return price - discount
def check_eligibility(age):
    """
    Determines whether a person is eligible to vote
    based on the minimum age requirement of 18 years.
    """
    if age >= 18:
        return True
    else:
        return False
# Function Calls
final_price = calculate_discount(1000, 0.10)
eligibility = check_eligibility(20)
print("Final Price after Discount:", final_price)
print("Voting Eligibility:", eligibility)

Final Price after Discount: 900.0
Voting Eligibility: True
```

Explanation:

This task demonstrates how AI can enhance the quality of existing documentation by refining unclear or inconsistent comments. Instead of generating new code, the AI focuses on improving clarity, structure, and professional tone while preserving the original technical meaning. This helps maintain consistency across the codebase and improves readability for developers.

Task Description -5 (Documentation – Prompt Detail Impact Study)

Task:

Study the impact of prompt detail on AI-generated documentation quality.

Instructions:

Create two prompts: one brief and one detailed.

- Use both prompts to document the same Python function.
- Compare the generated outputs.

Expected Output -5:

A comparison table highlighting differences in completeness, clarity, and accuracy of documentation.

Prompt:

Write a clear and technically accurate docstring for the following Python function.

Code & Output:

```
def process_scores(scores):
    total = sum(scores)
    average = total / len(scores)

    if average >= 90:
        grade = "A"
    elif average >= 75:
        grade = "B"
    elif average >= 50:
        grade = "C"
    else:
        grade = "Fail"

    return average, grade

# Function Call
avg, grade = process_scores([85, 90, 88])

print("Average:", avg)
print("Grade:", grade)
```

```
Average: 87.66666666666667
Grade: B
```

Explanation:

This task demonstrates how the level of detail in a prompt affects the quality of AI-generated documentation. The brief prompt produces a minimal summary with limited explanation. In contrast, the detailed prompt generates structured, complete, and professional documentation including parameter and return descriptions.