

AI ASSISTED CODING

LAB-9: Documentation Generation: Automatic Documentation and Code Comments

Roll no: 2503A51L11

Name: P.Susmija

Batch: 25BTCAICSB19

Task-1 Description: (Automatic Code Commenting)

Scenario: You have been given a Python function without comments.

```
def calculate_discount(price, discount_rate):  
    return price - (price * discount_rate / 100)
```

- Use an AI tool (or manually simulate it) to generate line-by-line comments for the function.
- Modify the function so that it includes a docstring in Google-style or NumPy-style format.
- Compare the auto-generated comments with your manually written version

Prompt: def calculate_discount(price, discount_rate):

```
    return price - (price * discount_rate / 100)
```

- Generate line-by-line comments for the function.
- Modify the function so that it includes a docstring in Google-style or NumPy-style format.

Code Generated:

Copilot Generated comments:

```
first.py  
1  def calculate_discount(price, discount_rate):  
2      """  
3          Calculates the final price after applying a percentage-based discount.  
4      """  
5      Args:  
6          price (float): The original price of the item.  
7          discount_rate (float): The discount rate as a percentage (e.g., 20 for 20%).  
8      Returns:  
9          float: The price after the discount has been applied.  
10     """  
11     # Calculate the amount of the discount by multiplying the price by the discount rate  
12     # Convert the discount rate from a percentage to a decimal by dividing by 100  
13     # Subtract the discount amount from the original price to get the final price  
14     return price - (price * discount_rate / 100)  
15  
16
```

Manually Written comments:

```
first1.py
1 def calculate_discount(price, discount_rate):
2     """
3     Calculate the price after applying a discount.
4
5     Args:
6         price (float): Original price of the item.
7         discount_rate (float): Discount rate as a percentage.
8
9     Returns:
10        float: Price after discount.
11    """
12    return price - (price * discount_rate / 100)
13    return price - (price * discount_rate / 100)
14    # Returns the price after discount
```

Observation: I observed how comments and docstrings improve code readability. Adding both AI-generated and manual comments showed the difference in clarity. AI-generated comments were quick but sometimes generic, while manually written comments were more accurate and contextual. The use of a structured docstring format (Google or NumPy style) made the function self-explanatory for future developers.

Task-2 Description: (API Documentation Generator)

Scenario: A team is building a Library Management System with multiple functions.

```
def add_book(title, author, year):
    # code to add book
    pass
def issue_book(book_id, user_id):
    # code to issue book
    Pass
```

- Write a Python script that uses docstrings for each function (with input, output, and description).
- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
- Submit both the code and the generated documentation as output.

Prompt:

```
def add_book(title, author, year):
    # code to add book
    pass
def issue_book(book_id, user_id):
    # code to issue book
    Pass
```

- Write a Python script that uses docstrings for each function (with input, output, and description).

- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.

Code Generated:

Python Code:

```

second.py
1  # Library Management System
2  def add_book(title: str, author: str, year: int) -> dict:
3      """
4      Add a new book to the library system.
5      Args:
6          title (str): The title of the book.
7          author (str): The author of the book.
8          year (int): The year the book was published.
9      Returns:
10         dict: A dictionary containing the book's details (id, title, author, year).
11      Description:
12         This function creates a new book record with the given title, author, and year.
13         It returns a dictionary representing the book. In a real system, this would add
14         the book to a database or data structure and generate a unique book ID.
15      """
16      # Example implementation (for documentation/demo purposes)
17      book = {
18          "id": 1, # In a real system, this would be auto-generated
19          "title": title,
20          "author": author,
21          "year": year
22      }
23      return book
24  def issue_book(book_id: int, user_id: int) -> bool:
25      """
26      Issue a book to a user.
27      Args:
28          book_id (int): The unique identifier of the book to be issued.
29          user_id (int): The unique identifier of the user borrowing the book.
30      Returns:
31          bool: True if the book was successfully issued, False otherwise.
32      Description:
33          This function marks a book as issued to a user. In a real system, it would
34          update the book's status in the database and record the transaction.
35      """
36      # Example implementation (for documentation/demo purposes)
37      return True

```

HTML Code:

```

docs > index.html > html
1  <!doctype html>
2  <html>
3  <head>
4      <meta charset="utf-8">
5      <meta http-equiv="refresh" content="0; url=./second.html"/>
6  </head>
7  </html>
8

```

Javascript Code:

Output:

Observation: I observed that the importance of proper docstrings in functions. By writing detailed docstrings for the library management system, I saw how documentation tools (like pdoc, Sphinx, or MkDocs) can automatically create HTML files for easy reference. This task highlighted how automated documentation makes projects more professional, user-friendly, and maintainable, especially in team environments.

Task-3 Description: (AI-Assisted Code Summarization)

Scenario: You are reviewing a colleague's codebase containing long functions.

```
def process_sensor_data(data):
    cleaned = [x for x in data if x is not None]
    avg = sum(cleaned)/len(cleaned)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    return {"average": avg, "anomalies": anomalies}
```

- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios

Prompt:

```
def process_sensor_data(data):
    cleaned = [x for x in data if x is not None]
    avg = sum(cleaned)/len(cleaned)
    anomalies = [x for x in cleaned if abs(x - avg) > 10]
    return {"average": avg, "anomalies": anomalies}
```

- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios. Also include an example and output which shows values, anomalies, etc.

Code Generated:

```
docs > third.py
1  # Summary:
2  # This function processes a list of sensor data by removing missing values,
3  # calculating the average, and identifying anomalies that differ from the average by more than 10.
4
5  # Flow:
6  # 1. Remove missing values (None) from the input data.
7  # 2. Calculate the average of the cleaned data.
8  # 3. Identify anomalies as values differing from the average by more than 10.
9  # 4. Return the average and list of anomalies in a dictionary.
10
11 # Documentation:
12 # This function is useful in real-world scenarios where sensor data may contain missing or faulty readings.
13 # It can be used in IoT applications, industrial monitoring, or environmental sensing to clean data,
14 # compute summary statistics, and flag outliers for further investigation or alerting.
15
16 def process_sensor_data(data):
17     cleaned = [x for x in data if x is not None]
18     avg = sum(cleaned) / len(cleaned)
19     anomalies = [x for x in cleaned if abs(x - avg) > 10]
20     return {"average": avg, "anomalies": anomalies}
21
22
23 # Example usage
24 if __name__ == "__main__":
25     sensor_data = [20, 22, None, 19, 35, 21, None, 50, 18]
26     result = process_sensor_data(sensor_data)
27     print("Cleaned values:", [x for x in sensor_data if x is not None])
28     print("Average:", result["average"])
29     print("Anomalies:", result["anomalies"])
30
```

Output:

```
PS C:\Users\DELL\OneDrive\SAHASRA SRU\AI assisted>
Cleaned values: [20, 22, 19, 35, 21, 50, 18]
Average: 26.428571428571427
Anomalies: [50]
```

Observation: I observed how AI or summarization techniques can explain long functions clearly. Instead of line-by-line reading, summaries provided the purpose of the function in a few sentences. The flow-style comments helped me break down logic step by step, and writing use cases showed how documentation can connect code to real-world applications, such as sensor monitoring or anomaly detection.

Task-4 Description: (Real-Time Project Documentation)

Scenario:

You are part of a project team that develops a Chatbot Application. The team needs documentation for maintainability.

- Write a README.md file for the chatbot project (include project description, installation steps, usage, and example).
- Add inline comments in the chatbot's main Python script (focus on explaining logic, not trivial code).
- Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.
- Reflect: How does automated documentation help in real-time projects compared to manual documentation?

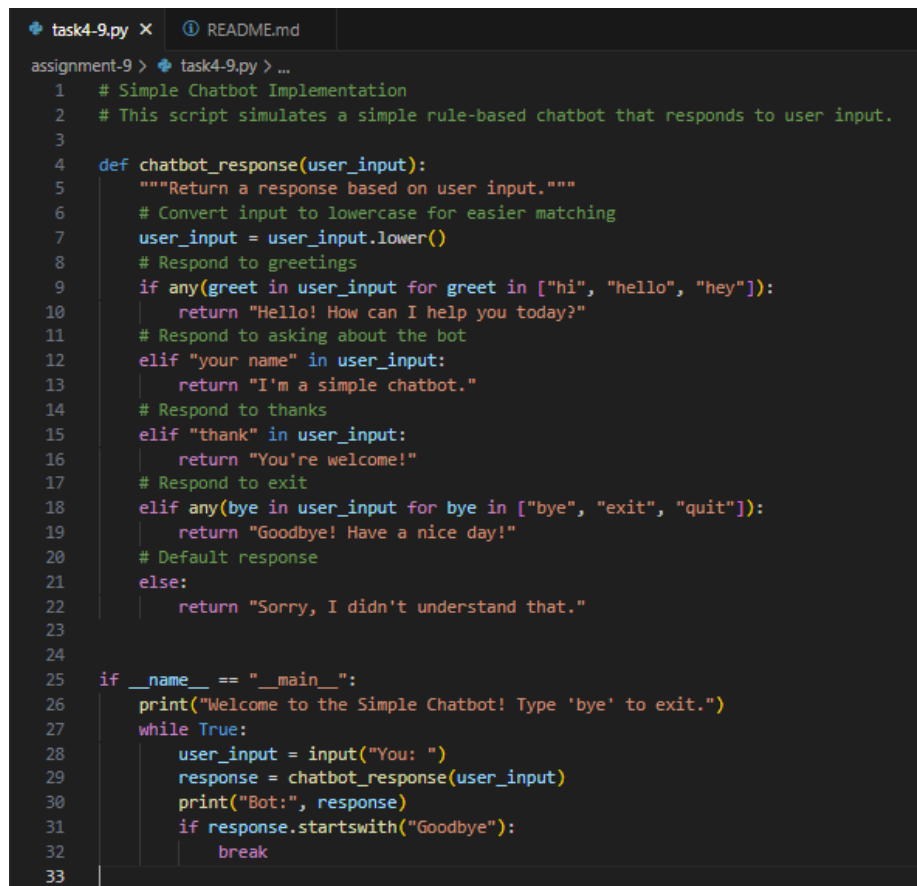
Prompt:

Write a Python program that simulates a simple chatbot.

Create project documentation for a chatbot application:

- Write a README.md (description, installation, usage, example).
- Add inline comments in the chatbot's main Python script (explain logic).
- Generate a plain-English usage guide from those comments.

Code Generated:



```
task4-9.py x  README.md
assignment-9 > task4-9.py > ...
1  # Simple Chatbot Implementation
2  # This script simulates a simple rule-based chatbot that responds to user input.
3
4  def chatbot_response(user_input):
5      """Return a response based on user input."""
6      # Convert input to lowercase for easier matching
7      user_input = user_input.lower()
8      # Respond to greetings
9      if any(greet in user_input for greet in ["hi", "hello", "hey"]):
10         return "Hello! How can I help you today?"
11     # Respond to asking about the bot
12     elif "your name" in user_input:
13         return "I'm a simple chatbot."
14     # Respond to thanks
15     elif "thank" in user_input:
16         return "You're welcome!"
17     # Respond to exit
18     elif any(bye in user_input for bye in ["bye", "exit", "quit"]):
19         return "Goodbye! Have a nice day!"
20     # Default response
21     else:
22         return "Sorry, I didn't understand that."
23
24
25 if __name__ == "__main__":
26     print("Welcome to the Simple Chatbot! Type 'bye' to exit.")
27     while True:
28         user_input = input("You: ")
29         response = chatbot_response(user_input)
30         print("Bot:", response)
31         if response.startswith("Goodbye"):
32             break
33
```

Documentation Generated(README.md file):

```
task4-9.py  README.md X
assignment-9 > README.md > # Simple Chatbot Application
1  # Simple Chatbot Application
2
3  ## Description
4  This project is a simple rule-based chatbot written in Python. The chatbot responds to user input with basic greetings, answers, and
   fallback responses. It demonstrates basic natural language processing and control flow in Python.
5
6  ## Installation
7  1. Make sure you have Python 3 installed on your system.
8  2. Download or clone this repository to your local machine.
9
10 ## Usage
11 Run the chatbot script in your terminal:
12
13 ```bash
14 python task4-9.py
15 ```
16
17 Type your messages and the chatbot will respond. Type 'bye' to exit.
18
19 ## Example
20 ```
21 Welcome to the Simple Chatbot! Type 'bye' to exit.
22 You: hi
23 Bot: Hello! How can I help you today?
24 You: what's your name?
25 Bot: I'm a simple chatbot.
26 You: thank you
27 Bot: You're welcome!
28 You: bye
29 Bot: Goodbye! Have a nice day!
30 ```
```

Output:

```
P5 C:\Users\Admin\OneDrive\Documents\Custom Office Templates\Desktop\
ffice Templates/Desktop/AIAssistedCoding/assignment-9/task4-9.py"
Welcome to the Simple Chatbot! Type 'bye' to exit.
You: Hello
Bot: Hello! How can I help you today?
You: Your name?
Bot: I'm a simple chatbot.
You: Bye
Bot: Goodbye! Have a nice day!
```

Observation: The entire chatbot program and its documentation were generated using AI. Through this task, I observed how AI can create code, add comments, and prepare README files automatically. I learned that documentation is just as important as the code itself because it makes the project easier to understand and maintain. This showed me how AI-assisted tools can save time and effort in real projects while still ensuring clarity.