AI ASSISTED CODING

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

Roll no: 2503A51L39

Name: shaik faheem

Batch: 25BTCAICSB20

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

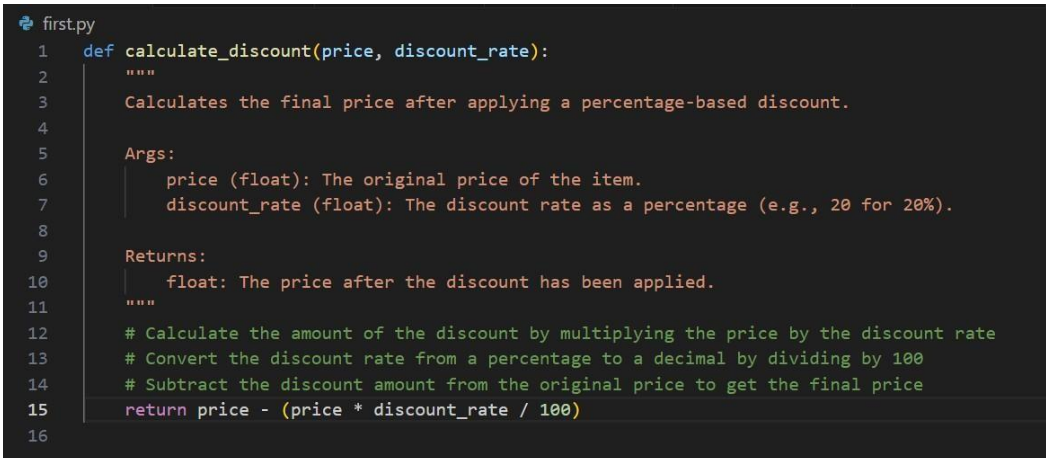
Prompt: def calculate_discount(price, discount_rate):

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 NumPystyle format.

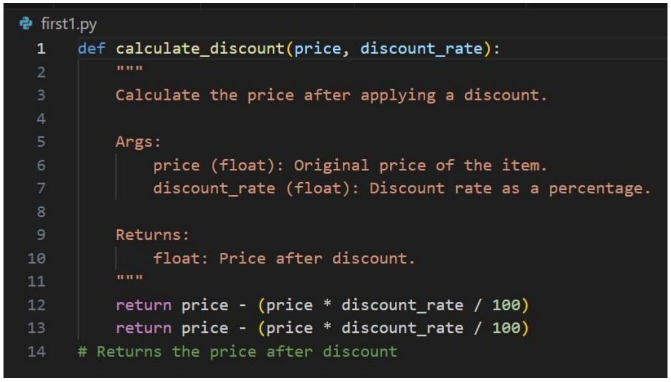
Code Generated:

Copilot Generated comments:



```python
def calculate_discount(price, discount_rate):
    """
    Calculates the final price after applying a percentage-based discount.

    Args:
        price (float): The original price of the item.
        discount_rate (float): The discount rate as a percentage (e.g., 20 for 20%).

    Returns:
        float: The price after the discount has been applied.
    """
    # Calculate the amount of the discount by multiplying the price by the discount rate
    # Convert the discount rate from a percentage to a decimal by dividing by 100
    # Subtract the discount amount from the original price to get the final price
    return price - (price * discount_rate / 100)

```

Manually Written comments:



```python
def calculate_discount(price, discount_rate):
    """
    Calculate the price after applying a discount.

    Args:
        price (float): Original price of the item.
        discount_rate (float): Discount rate as a percentage.

    Returns:
        float: Price after discount.
    """
    return price - (price * discount_rate / 100)
    return price - (price * discount_rate / 100)
# 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.

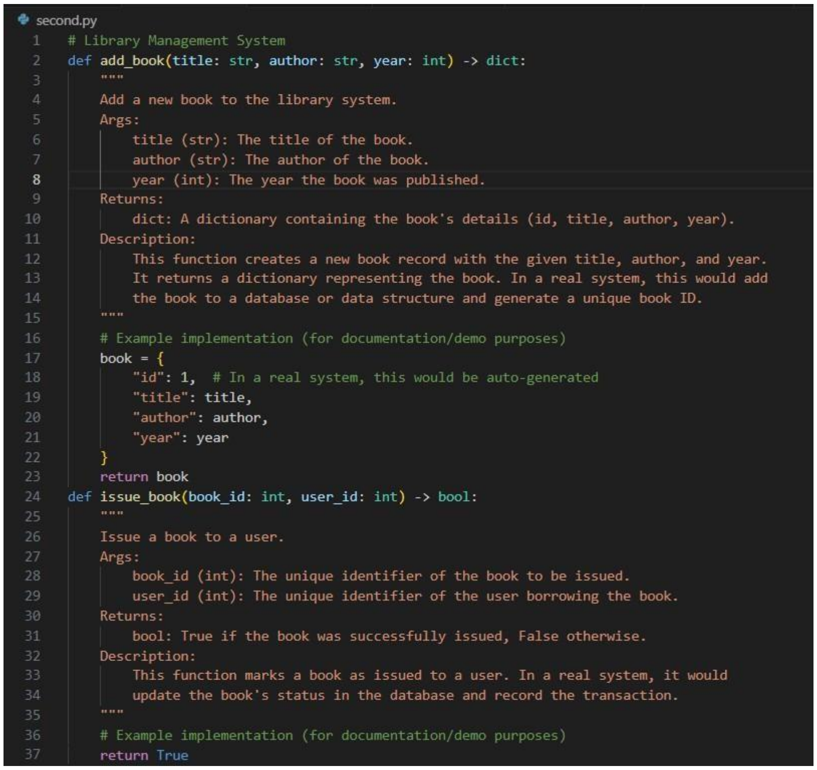
Prompt:

```
        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.

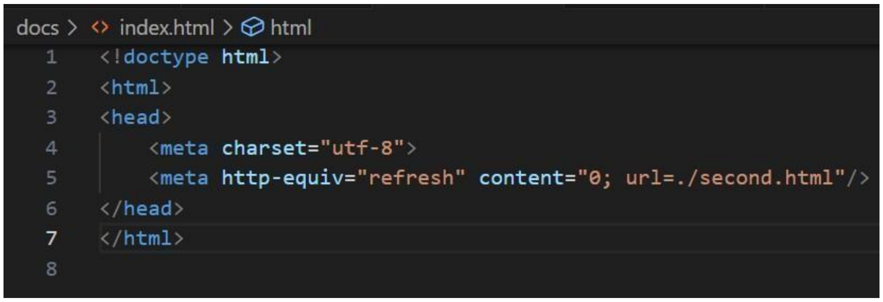
Code Generated:

Python Code:

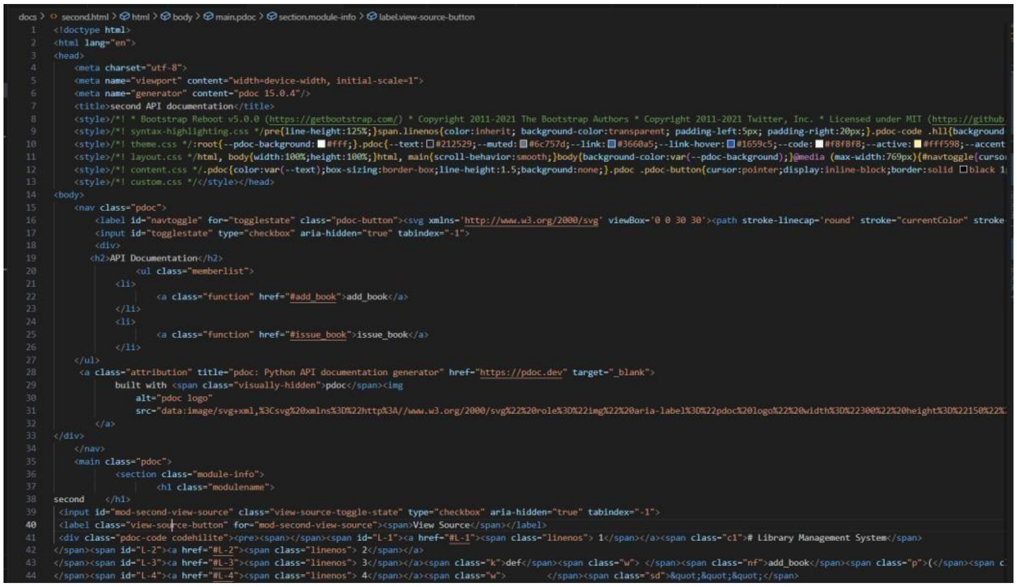


```python
# Library Management System
def add_book(title: str, author: str, year: int) -> dict:
    """
    Add a new book to the library system.
    Args:
        title (str): The title of the book.
        author (str): The author of the book.
        year (int): The year the book was published.
    Returns:
        dict: A dictionary containing the book's details (id, title, author, year).
    Description:
        This function creates a new book record with the given title, author, and year.
        It returns a dictionary representing the book. In a real system, this would add
        the book to a database or data structure and generate a unique book ID.
    """
    # Example implementation (for documentation/demo purposes)
    book = {
        "id": 1,  # In a real system, this would be auto-generated
        "title": title,
        "author": author,
        "year": year
    }
    return book
def issue_book(book_id: int, user_id: int) -> bool:
    """
    Issue a book to a user.
    Args:
        book_id (int): The unique identifier of the book to be issued.
        user_id (int): The unique identifier of the user borrowing the book.
    Returns:
        bool: True if the book was successfully issued, False otherwise.
    Description:
        This function marks a book as issued to a user. In a real system, it would
        update the book's status in the database and record the transaction.
    """
    # Example implementation (for documentation/demo purposes)
    return True
```

HTML Code:

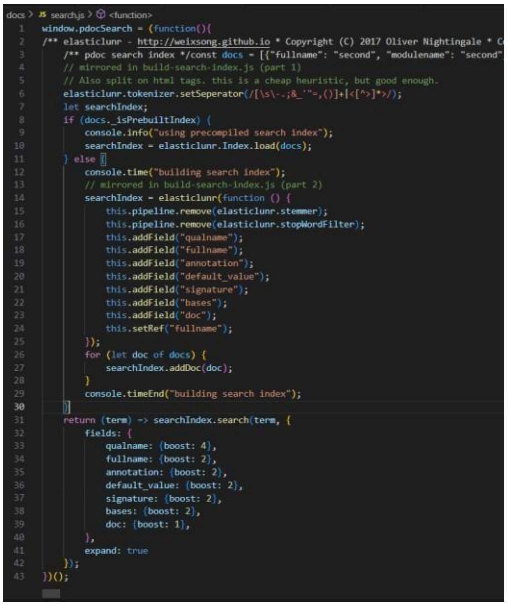


```html
<!doctype html>
<html>
<head>
    <meta charset="utf-8">
    <meta http-equiv="refresh" content="0; url=./second.html"/>
</head>
</html>

```


docs > second.html > html > body > main.pdoc > section.module-info > label.view-source-button

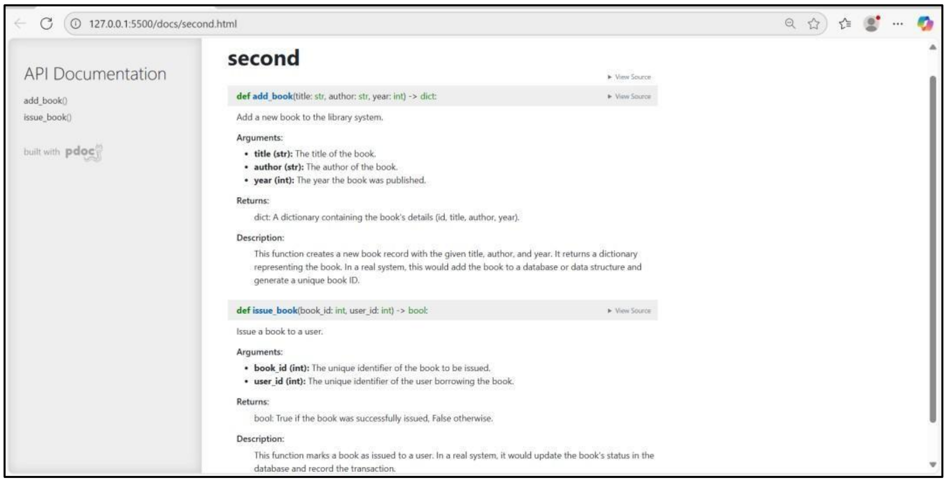
```html
<!doctype html>
<html lang="en">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta name="generator" content="pdoc 15.0.4"/>
    <title>second API documentation</title>
    <style>/*! * Bootstrap Reboot v5.0.0 (https://getbootstrap.com/) * Copyright 2011-2021 The Bootstrap Authors * Copyright 2011-2021 Twitter, Inc. * Licensed under MIT (https://github
    <style>/*! syntax-highlighting.css */pre{line-height:125%;}span.linenos{color:inherit; background-color:transparent; padding-left:5px; padding-right:20px;}.pdoc-code .hll{background
    <style>/*! theme.css */:root{--pdoc-background:#fff;}.pdoc{--text:#212529;--muted:#6c757d;--link:#3660a5;--link-hover:#1659c5;--code:#f8f8f8;--active:#fff598;--accent
    <style>/*! layout.css */html, body{width:100%;height:100%;}html, main{scroll-behavior:smooth;}body{background-color:var(--pdoc-background);}@media (max-width:769px){#navtoggle{curso
    <style>/*! content.css */.pdoc{color:var(--text);box-sizing:border-box;line-height:1.5;background:none;}.pdoc .pdoc-button{cursor:pointer;display:inline-block;border:solid black 1
    <style>/*! custom.css */</style></head>
<body>
    <nav class="pdoc">
        <label id="navtoggle" for="togglestate" class="pdoc-button"><svg xmlns='http://www.w3.org/2000/svg' viewBox='0 0 30 30'><path stroke-linecap='round' stroke="currentColor" stroke
        <input id="togglestate" type="checkbox" aria-hidden="true" tabindex="-1">
        <div>
       <h2>API Documentation</h2>
                <ul class="memberlist">
            <li>
                    <a class="function" href="#add_book">add_book</a>
            </li>
            <li>
                    <a class="function" href="#issue_book">issue_book</a>
            </li>
    </ul>
     <a class="attribution" title="pdoc: Python API documentation generator" href="https://pdoc.dev" target="_blank">
            built with <span class="visually-hidden">pdoc</span><img
                alt="pdoc logo"
                src="data:image/svg+xml,%3Csvg%20xmlns%3D%22http%3A//www.w3.org/2000/svg%22%20role%3D%22img%22%20aria-label%3D%22pdoc%20logo%22%20width%3D%22300%22%20height%3D%22150%22%
        </a>
</div>
    </nav>
    <main class="pdoc">
            <section class="module-info">
                    <h1 class="modulename">
second    </h1>
 <input id="mod-second-view-source" class="view-source-toggle-state" type="checkbox" aria-hidden="true" tabindex="-1">
 <label class="view-source-button" for="mod-second-view-source"><span>View Source</span></label>
 <div class="pdoc-code codehilite"><pre><span></span><span id="L-1"><a href="#L-1"><span class="linenos"> 1</span></a><span class="c1"># Library Management System</span>
</span><span id="L-2"><a href="#L-2"><span class="linenos"> 2</span></a>
</span><span id="L-3"><a href="#L-3"><span class="linenos"> 3</span></a><span class="k">def</span><span class="w"> </span><span class="nf">add_book</span><span class="p">(</span><span c
</span><span id="L-4"><a href="#L-4"><span class="linenos"> 4</span></a><span class="w">        </span><span class="sd">&quot;&quot;&quot;</span>
```

Javascript Code:



```javascript
window.pdocSearch = (function(){
/** elasticlunr - http://weixsong.github.io * Copyright (C) 2017 Oliver Nightingale * C
    /** pdoc search index */const docs = [{"fullname": "second", "modulename": "second"
    // mirrored in build-search-index.js (part 1)
    // Also split on html tags. this is a cheap heuristic, but good enough.
    elasticlunr.tokenizer.setSeperator(/[\s\-.;&_'"=,()]+|<[^>]*>/);
    let searchIndex;
    if (docs._isPrebuiltIndex) {
        console.info("using precompiled search index");
        searchIndex = elasticlunr.Index.load(docs);
    } else {
        console.time("building search index");
        // mirrored in build-search-index.js (part 2)
        searchIndex = elasticlunr(function () {
            this.pipeline.remove(elasticlunr.stemmer);
            this.pipeline.remove(elasticlunr.stopWordFilter);
            this.addField("qualname");
            this.addField("fullname");
            this.addField("annotation");
            this.addField("default_value");
            this.addField("signature");
            this.addField("bases");
            this.addField("doc");
            this.setRef("fullname");
        });
        for (let doc of docs) {
            searchIndex.addDoc(doc);
        }
        console.timeEnd("building search index");
    }
    return (term) => searchIndex.search(term, {
        fields: {
            qualname: {boost: 4},
            fullname: {boost: 2},
            annotation: {boost: 2},
            default_value: {boost: 2},
            signature: {boost: 2},
            bases: {boost: 2},
            doc: {boost: 1},
        },
        expand: true
    });
})();
```

Output:



second

def add_book(title: str, author: str, year: int) -> dict:

Add a new book to the library system.

Arguments:

- **title (str):** The title of the book.
- **author (str):** The author of the book.
- **year (int):** The year the book was published.

Returns:

dict: A dictionary containing the book's details (id, title, author, year).

Description:

This function creates a new book record with the given title, author, and year. It returns a dictionary representing the book. In a real system, this would add the book to a database or data structure and generate a unique book ID.

def issue_book(book_id: int, user_id: int) -> bool:

Issue a book to a user.

Arguments:

- **book_id (int):** The unique identifier of the book to be issued.
- **user_id (int):** The unique identifier of the user borrowing the book.

Returns:

bool: True if the book was successfully issued, False otherwise.

Description:

This function marks a book as issued to a user. In a real system, it would update the book's status in the database and record the transaction.

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

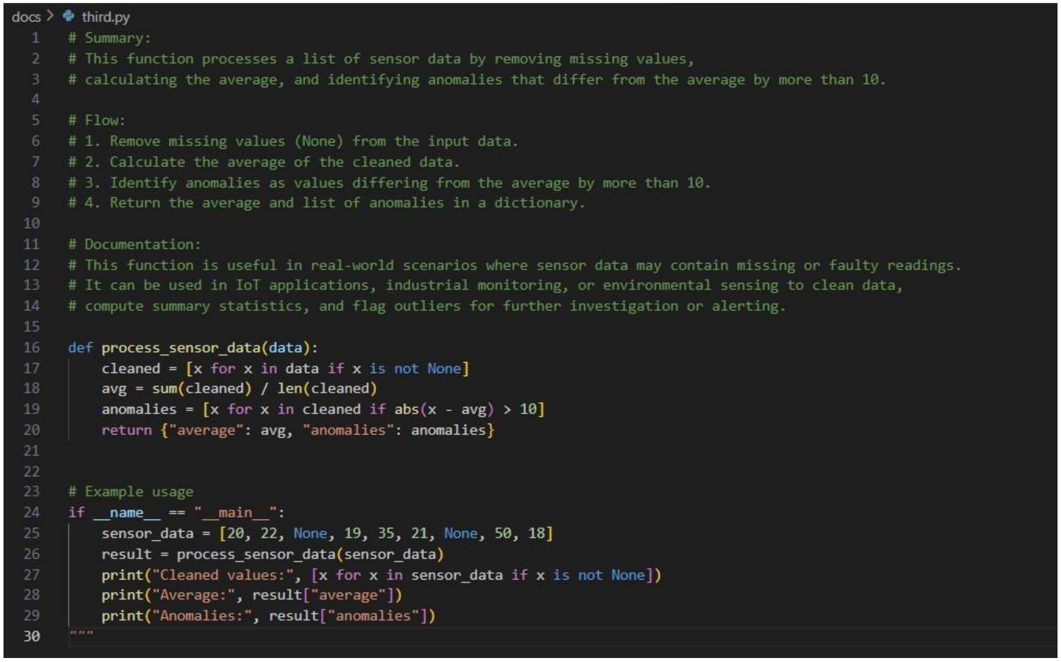
Prompt:

```
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}
```

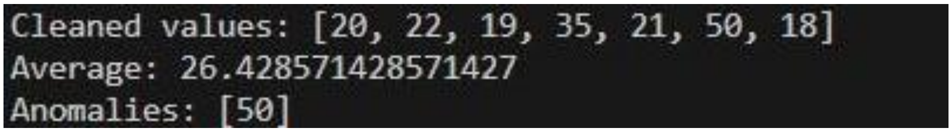
- Generate a summary comment explaining the purpose of the function in 2–3 lines.
- Create a flow-style comment (step-by-step explanation).
- Write a short paragraph of documentation describing possible use cases of this function in real-world scenarios. Also include a example and ouput which shows values,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

Code Generated:



```python
# Summary:
# This function processes a list of sensor data by removing missing values,
# calculating the average, and identifying anomalies that differ from the average by more than 10.

# Flow:
# 1. Remove missing values (None) from the input data.
# 2. Calculate the average of the cleaned data.
# 3. Identify anomalies as values differing from the average by more than 10.
# 4. Return the average and list of anomalies in a dictionary.

# Documentation:
# This function is useful in real-world scenarios where sensor data may contain missing or faulty readings.
# It can be used in IoT applications, industrial monitoring, or environmental sensing to clean data,
# compute summary statistics, and flag outliers for further investigation or alerting.

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}


# Example usage
if __name__ == "__main__":
    sensor_data = [20, 22, None, 19, 35, 21, None, 50, 18]
    result = process_sensor_data(sensor_data)
    print("Cleaned values:", [x for x in sensor_data if x is not None])
    print("Average:", result["average"])
    print("Anomalies:", result["anomalies"])
"""
```

Output:



```
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 online-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).

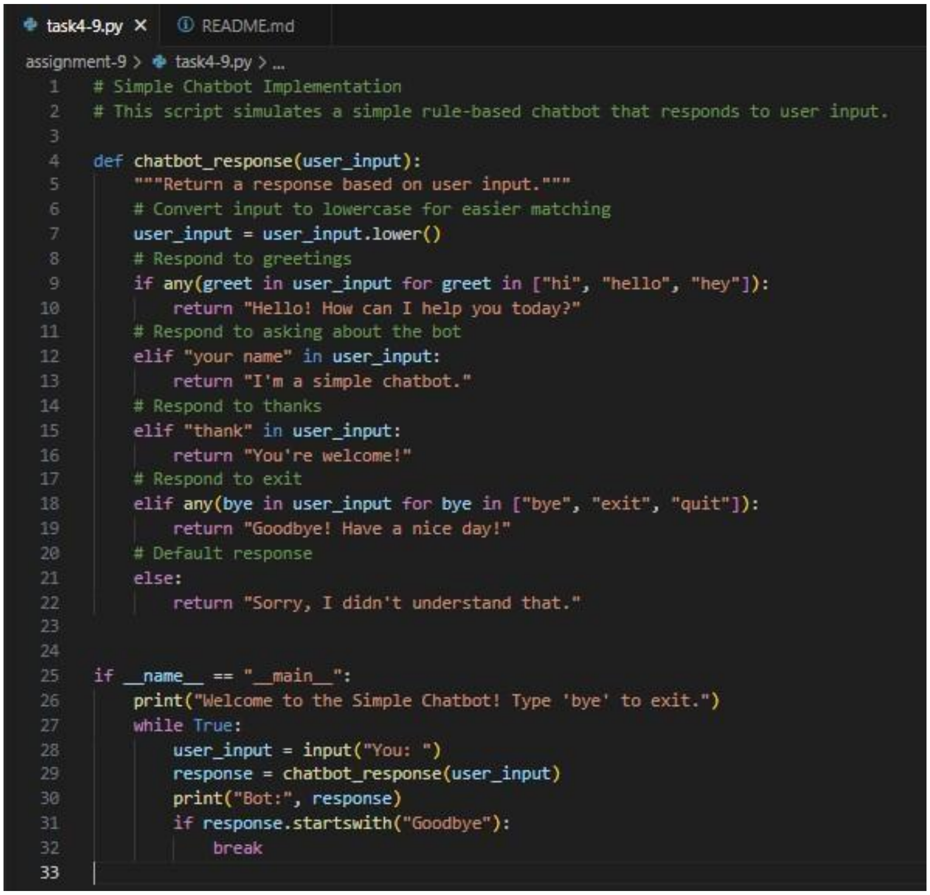
- Add inline comments in the chatbot's main Python script (focus on explaining logic, not trivial code).
- Use an AI-assisted tool (or simulate it) to generate a usage guide in plain English from your code comments.
- Reflect: How does automated documentation help in real-time projects compared to manual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

Prompt:

Write a Python program that simulates a simple chatbot.
Create project documentation for a chatbot application:

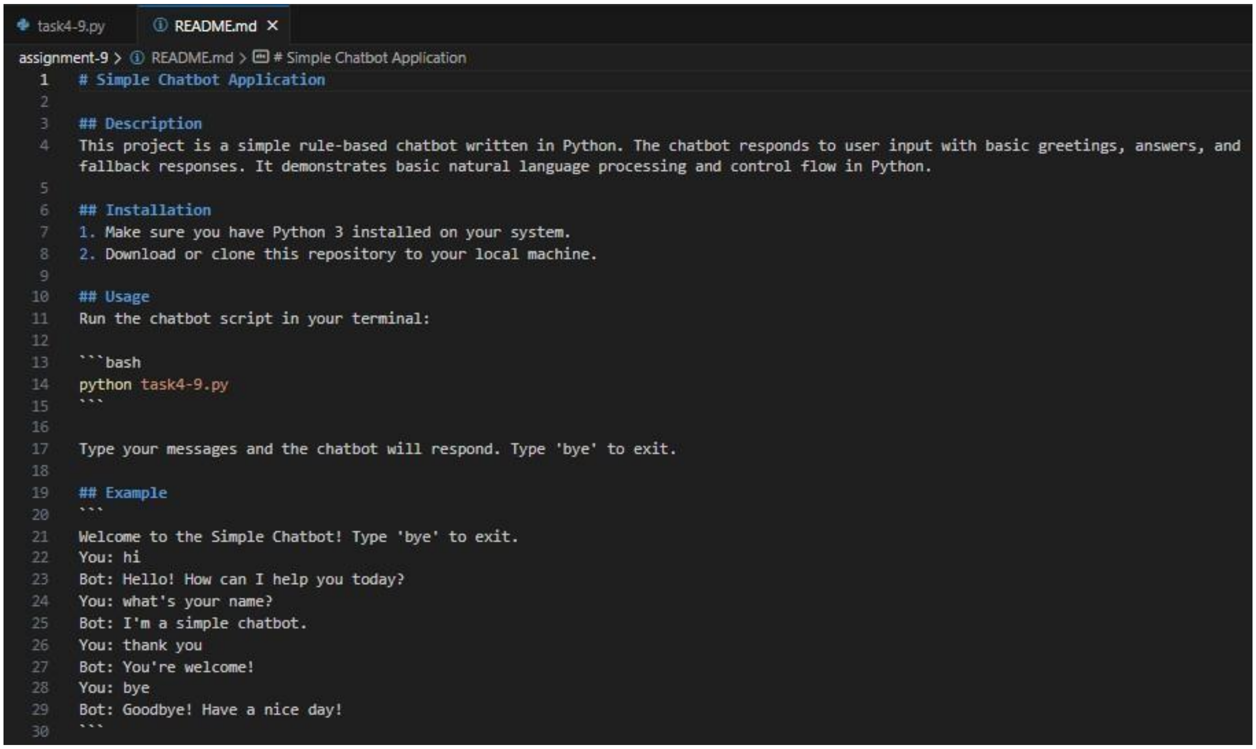
- Write a README.md (description, installation, usage, example).
- Add inline comments in the chatbot's main Python script (explain logic).
- Generate a plain-English usage guide from those comments. **Code Generated:**

Output:



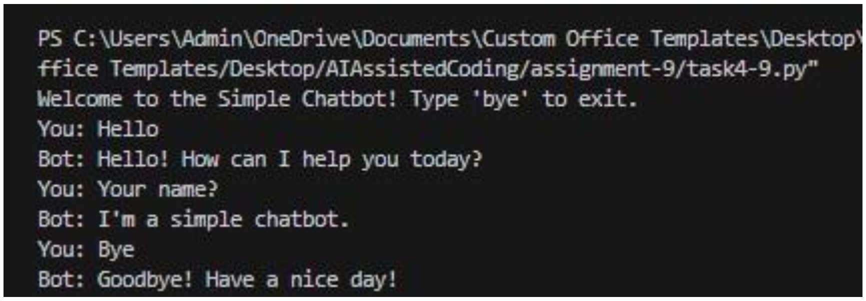
```python
# Simple Chatbot Implementation
# This script simulates a simple rule-based chatbot that responds to user input.

def chatbot_response(user_input):
    """Return a response based on user input."""
    # Convert input to lowercase for easier matching
    user_input = user_input.lower()
    # Respond to greetings
    if any(greet in user_input for greet in ["hi", "hello", "hey"]):
        return "Hello! How can I help you today?"
    # Respond to asking about the bot
    elif "your name" in user_input:
        return "I'm a simple chatbot."
    # Respond to thanks
    elif "thank" in user_input:
        return "You're welcome!"
    # Respond to exit
    elif any(bye in user_input for bye in ["bye", "exit", "quit"]):
        return "Goodbye! Have a nice day!"
    # Default response
    else:
        return "Sorry, I didn't understand that."


if __name__ == "__main__":
    print("Welcome to the Simple Chatbot! Type 'bye' to exit.")
    while True:
        user_input = input("You: ")
        response = chatbot_response(user_input)
        print("Bot:", response)
        if response.startswith("Goodbye"):
            break

```

Documentation Generated(README.md file):



~~~markdown
# Simple Chatbot Application

## Description
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.

## Installation
1. Make sure you have Python 3 installed on your system.
2. Download or clone this repository to your local machine.

## Usage
Run the chatbot script in your terminal:

```bash
python task4-9.py
```


Type your messages and the chatbot will respond. Type 'bye' to exit.

## Example
```
Welcome to the Simple Chatbot! Type 'bye' to exit.
You: hi
Bot: Hello! How can I help you today?
You: what's your name?
Bot: I'm a simple chatbot.
You: thank you
Bot: You're welcome!
You: bye
Bot: Goodbye! Have a nice day!
```
~~~

Output:



```
PS 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.