SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE		DEPARTMENT OF COMPUTER SCIENCE ENGINEERING	
Program Name: B. Tech		**Assignment Type: Lab**	**Academic Year:**2025-2026
Course Coordinator Name	Venkataramana Veeramsetty		
Instructor(s) Name	Dr. V. Venkataramana (Co-ordinator) Dr. T. Sampath Kumar Dr. Pramoda Patro Dr. Brij Kishor Tiwari Dr.J.Ravichander Dr. Mohammand Ali Shaik Dr. Anirodh Kumar Mr. S.Naresh Kumar Dr. RAJESH VELPULA Mr. Kundhan Kumar Ms. Ch.Rajitha Mr. M Prakash Mr. B.Raju Intern 1 (Dharma teja) Intern 2 (Sai Prasad) Intern 3 (Sowmya) NS_2 (Mounika)		
Course Code	24CS002PC215	**Course Title**	AI Assisted Coding
Year/Sem	II/I	**Regulation**	R24
Date and Day of Assignment	Week5 - Monday	**Time(s)**	
Duration	2 Hours	**Applicable to Batches**	
AssignmentNumber: 9.1(Present assignment number)/**24**(Total number of assignments)			

Q.No.	Question	*Expected Time to complete*
1	**Lab 9 – Documentation Generation: Automatic Documentation and Code Comments** **Lab Objectives** • Inline comments • Docstrings • Auto-documentation tools • AI-assisted summarization	Week5 - Monday

Task Description #1 (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.

Aigenertedcode:

```
import numpy as np
import IPython.display as display
from matplotlib import pyplot as plt
import io
import base64
ys = 200 + np.random.randn(100)
x = [x for x in range(len(ys))]
fig = plt.figure(figsize=(4, 3), facecolor='w')
plt.plot(x, ys, '-')
plt.fill_between(x, ys, 195, where=(ys > 195), facecolor='g', alpha=0.6)
plt.title("Sample Visualization", fontsize=10)
data = io.BytesIO()
plt.savefig(data)
image = F"{base64.b64encode(data.getvalue()).decode()}"
alt = "Sample Visualization"
display.display(display.Markdown(F"""![{alt}]({image})"""))
plt.close(fig)
def calculate_discount(price, discount_rate):
    discount_amount = price * discount_rate / 100
    return price - discount_amount
```

Manually return code:

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

    Args:
        price (float): The original price.
        discount_rate (float): The discount rate as a percentage.

    Returns:
        float: The final price after applying the discount.
    """
    return price - (price * discount_rate / 100)

# Example usage:
print(calculate_discount(100, 20))  # Output: 80.0
print(calculate_discount(250, 10))  # Output: 225.0
```

```
80.0
225.0
```

Command prompt:

1. "Add line-by-line comments as an AI tool might generate them to explain what each line does."

2. *"Write manual comments as if you're explaining the code to another developer or a beginner."*

3. "Add a Google-style docstring to explain what the function does, its arguments, and its return value."

4. "Replace the docstring with a NumPy-style version for the same function."

5. "Write a short comparison between the AI-generated and manual comments. Which is clearer or more informative?"

Comparision of Ai Generated code and Manually return code:

Here's a Comparison Table: AI-Generated vs Manual (Docstring) Version

Aspect	AI-Generated Comments (Inline)	Manually Written Docstring (Google-style)
Location	Inline (before/inside function lines)	Inside a multi-line string under the function definition
Clarity	Describes what each line is doing	Describes the entire function's purpose, inputs, and output
Readability	Easier to follow line-by-line	Easier to read and understand overall intent at a glance
Standardization	Not standardized	Follows Google-style docstring format (standard in many codebases)
Tooling Support	Rarely picked up by documentation tools	Recognized by tools like pdoc, Sphinx, PyCharm, etc.
Scalability	Not ideal for large or complex functions	Scales well with more complex or parameter-heavy functions
Use Case	Great for beginner learning or internal debugging	Great for professional documentation, API reference, and team collaboration
Example Output Included	Usually not included	Can be included after the docstring or in the docstring as examples

Task Description #2 (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).

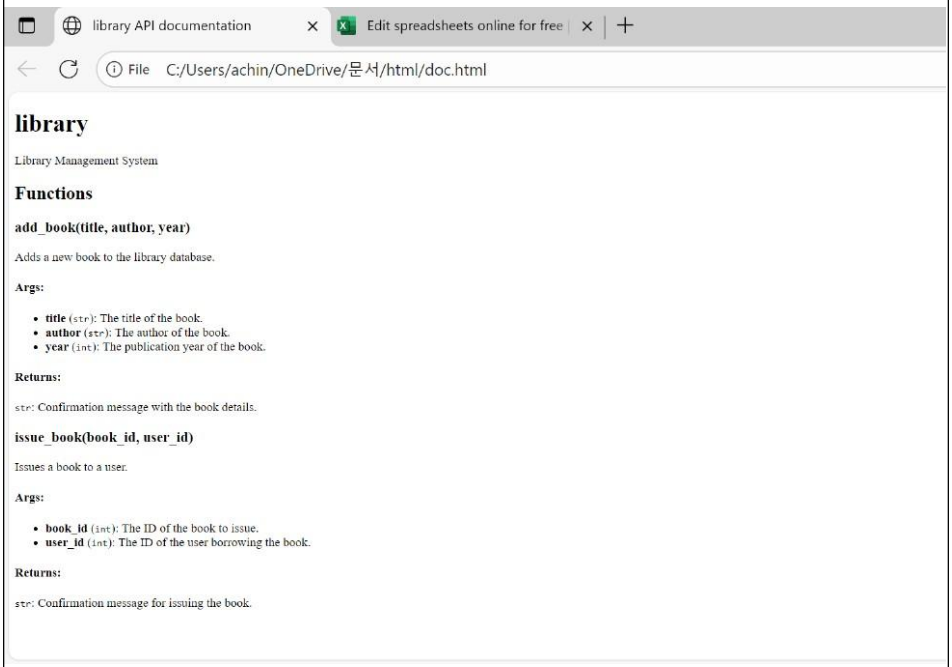
- Use a documentation generator tool (like pdoc, Sphinx, or MkDocs) to automatically create HTML documentation.
- Submit both the code and the generated documentation as output.

Python Generated code and output:

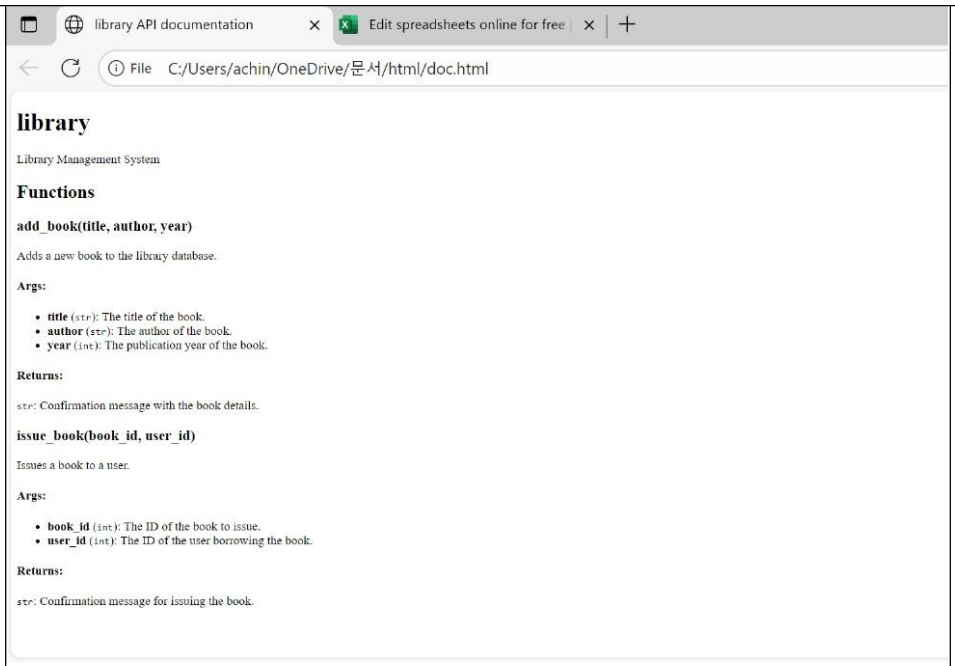
```
    return f"Book added: '{title}' by {author} ({year})"
def issue_book(book_id, user_id):
    return f"Book ID {book_id} issued to User ID {user_id}"
if __name__ == "__main__":
    print(add_book("The Alchemist", "Paulo Coelho", 1988))
    print(issue_book(102, 1002))
```

```
Book added: 'The Alchemist' by Paulo Coelho (1988)
Book ID 102 issued to User ID 1002
```

HTML Generated Code:



OUTPUT:



Task Description #3 (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.

Flow style comment:

```python
def process_sensor_data(data):
    # Step 1: Remove None values from the dataset
    cleaned = [x for x in data if x is not None]

    # Step 2: Calculate the average of the cleaned data
    avg = sum(cleaned) / len(cleaned)

    # Step 3: Identify values that differ from the average by more than 10 units
    anomalies = [x for x in cleaned if abs(x - avg) > 10]

    # Step 4: Return a dictionary with the average and list of anomalies
    return {"average": avg, "anomalies": anomalies}
```

A Paragraph above documentation describing possible uses cases of this function in real world senarios:

This function can be used in real-time monitoring systems where sensor data is continuously collected — for example, in IoT devices, industrial machinery, or environmental monitoring. It helps detect abnormal readings that might indicate hardware issues, environmental changes, or system faults. By filtering out invalid data and identifying significant deviations, it can serve as a lightweight anomaly detection mechanism in edge computing scenarios or data preprocessing pipelines.

Task Description #4 (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?

CODE:

```python
# chatbot.py

def preprocess_input(user_input):
    return user_input.lower().strip()
def generate_response(cleaned_input):
    if "hello" in cleaned_input or "hi" in cleaned_input:
        return "Hi there! How can I help you today?"
    elif "weather" in cleaned_input:
        return "I'm sorry, I don't have weather data right now. Can I help with anything else?"
    else:
        return "I'm not sure how to answer that. Could you try asking something else?"
def main():
    print("Welcome to the Chatbot! Type 'exit' to quit.")
    while True:
        user_input = input("User: ")
        if user_input.lower() == "exit":
            print("Chatbot: Goodbye!")
            break
        cleaned_input = preprocess_input(user_input)
        response = generate_response(cleaned_input)
        print(f"Chatbot: {response}")

if __name__ == "__main__":
    main()
```

OUTPUT:

```
Welcome to the Chatbot! Type 'exit' to quit.
User: what is the weather today?
Chatbot: I'm sorry, I don't have weather data right now. Can I help with anything else?
User: hii
Chatbot: Hi there! How can I help you today?
User:
```

How does automated documentation help in real-time projects compared to manual documentation

Automated Documentation Advantages:
Always up-to-date: Generated directly from code, so no outdated info.
Saves time: No need to write separate docs manually.
Consistent format: Ensures uniform, clear documentation.
Boosts productivity: Easy access to accurate docs speeds up development.
Better onboarding: New team members understand code faster.

Manual Documentation Drawbacks:
Can become outdated quickly.
Time-consuming to write and maintain.
Inconsistent quality and style.