

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE			DEPARTMENT OF COMPUTER SCIENCE ENGINEERING		
ProgramName:B. Tech			Assignment Type: Lab	AcademicYear:2025-2026	
CourseCoordinatorName			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)		
CourseCode		24CS002PC215	CourseTitle	AI Assisted Coding	
Year/Sem		II/I	Regulation	R24	
Date and Day of Assignment		Week4 - Wednesday	Time(s)		
Duration		2 Hours	Applicableto Batches		
AssignmentNumber:9.3(Present assignment number)/24(Total number of assignments)					
Q.No.	Question				ExpectedTime to complete
1	<div>Lab 8: Documentation Generation: Automatic documentation and code comments</div> <div>Lab Objectives:</div> <div><ul style="list-style-type: none">To understand the importance of documentation and code comments in software development.To explore how AI-assisted coding tools can generate meaningful documentation and inline comments.To practice generating function-level and module-level docstrings automatically.To evaluate the quality, accuracy, and limitations of AI-generated documentation.To develop a small automated tool for documentation generation in Python..</div>				Week4 - Wednesday

Lab Outcomes (LOs):

After completing this lab, students will be able to:

- Apply AI-assisted coding tools to generate docstrings and inline comments for Python code.
- Critically analyze AI-generated documentation for correctness, completeness, and readability.
- Create structured documentation (function-level, module-level) following standard formats.
- Design and implement a mini documentation generator tool to automate code commenting and docstring creation.

Task Description#1 Basic Docstring Generation

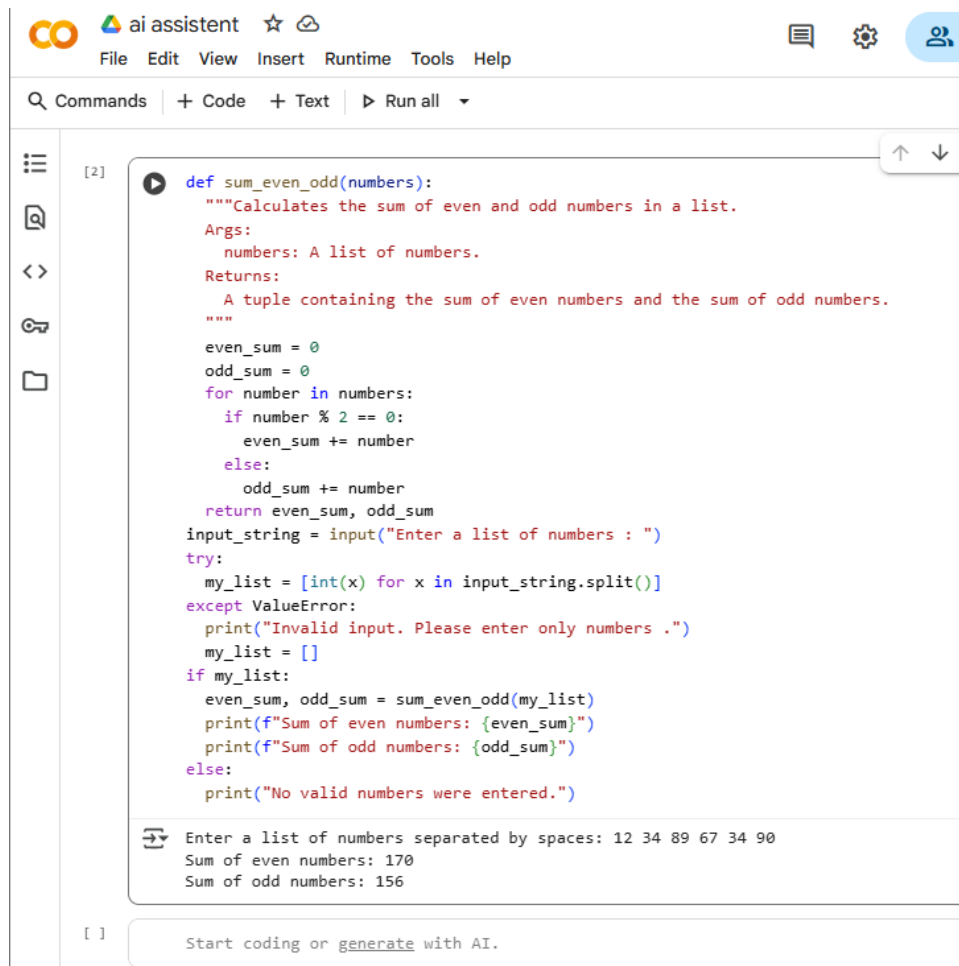
- Write python function to return sum of even and odd numbers in the given list.
- Incorporate manual **docstring** in code with Google Style
- Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
- Compare the AI-generated docstring with your manually written one.

Expected Outcome#1: Students understand how AI can produce function-level documentation.

PROMPT :

- write a python function code to return sum of even and odd number in the given list by using the docstring with google style.

CODE :



```
def sum_even_odd(numbers):
    """Calculates the sum of even and odd numbers in a list.
    Args:
        numbers: A list of numbers.
    Returns:
        A tuple containing the sum of even numbers and the sum of odd numbers.
    """
    even_sum = 0
    odd_sum = 0
    for number in numbers:
        if number % 2 == 0:
            even_sum += number
        else:
            odd_sum += number
    return even_sum, odd_sum

input_string = input("Enter a list of numbers : ")
try:
    my_list = [int(x) for x in input_string.split()]
except ValueError:
    print("Invalid input. Please enter only numbers .")
    my_list = []
if my_list:
    even_sum, odd_sum = sum_even_odd(my_list)
    print(f"Sum of even numbers: {even_sum}")
    print(f"Sum of odd numbers: {odd_sum}")
else:
    print("No valid numbers were entered.")
```

Enter a list of numbers separated by spaces: 12 34 89 67 34 90
Sum of even numbers: 170
Sum of odd numbers: 156

Start coding or [generate](#) with AI.

OBSERVATION :

- the sum of the even numbers (12, 34, 34, 90) as 170 and the sum of the odd numbers (89, 67) as 156. The program handled the input string, converted it to a list of numbers, and then applied the `sum_even_odd` function as intended.

Task Description#2 Automatic Inline Comments

- Write python program for **sru_student** class with attributes like name, roll no., hostel_status and **fee_update** method and **display_details** method.
- Write comments manually for each line/code block
- Ask an AI tool to add inline comments explaining each line/step.
- Compare the AI-generated comments with your manually written one.

Expected Output#2: Students critically analyze AI-generated code comments.

PROMPT:

- "Write a Python program for an `sru_student` class with attributes like name, roll number, and hostel status. Add a `fee_update` method and a `display_details` method. Take dynamic input from the user for student details and fee, then display the updated details with proper inline comments."

CODE:

1. Manually with comments:

```
class sru_student:
    """Represents a student at SRU."""
    def __init__(self, name, roll_no, hostel_status): # Corrected the constructor name to __init__
        """Initializes a new sru_student object.
        Args:
            name: The student's name.
            roll_no: The student's roll number.
            hostel_status: The student's hostel status (e.g., 'resident', 'day scholar').
        """
        self.name = name
        self.roll_no = roll_no
        self.hostel_status = hostel_status
        self.fees_paid = 0 # Initialize fees_paid attribute

    #method for updating the student fee details
    def fee_update(self, amount):
        if amount > 0:
            self.fees_paid += amount
            print(f"Fee of {amount} updated for {self.name}. Total fees paid: {self.fees_paid}")
        else:
            print("Invalid amount. Fee amount must be positive.")

    #method to display the student's details
    def display_details(self):
        """Displays the student's details."""
        print("\nStudent Details:")
        print(f"Name: {self.name}")
        print(f"Roll No.: {self.roll_no}")
        print(f"Hostel Status: {self.hostel_status}")
        print(f"Fees Paid: {self.fees_paid}")
```



Example usage:

```
student1 = sru_student("Alice", "SRU123", "resident") # Corrected constructor call
student2 = sru_student("Bob", "SRU456", "day scholar") # Corrected constructor call
student1.display_details()
student2.display_details()
student1.fee_update(5000) # Updated fee_update call with amount
student2.fee_update(2000) # Updated fee_update call with amount
student1.fee_update(3000) # Updated fee_update call with amount
student1.display_details() #display student1 details
student2.display_details() #display student2 details
```



Student Details:

Name: Alice

Roll No.: SRU123

Hostel Status: resident

Fees Paid: 0

Student Details:

Name: Bob

Roll No.: SRU456

Hostel Status: day scholar

Fees Paid: 0

Fee of 5000 updated for Alice. Total fees paid: 5000

Fee of 2000 updated for Bob. Total fees paid: 2000

Fee of 3000 updated for Alice. Total fees paid: 8000

Student Details:

Name: Alice

Roll No.: SRU123

Hostel Status: resident

Fees Paid: 8000

Student Details:

Name: Bob

Roll No.: SRU456

Hostel Status: day scholar

Fees Paid: 2000

2. AI with comments:

```

class sru_student:
    """Represents a student at SRU.""" # Docstring explaining the purpose of the class

    def __init__(self, name, roll_no, hostel_status): # Constructor method to initialize a new student object
        """Initializes a new sru_student object.

        Args:
            name: The student's name.
            roll_no: The student's roll number.
            hostel_status: The student's hostel status (e.g., 'resident', 'day scholar').
        """ # Docstring explaining the constructor and its arguments
        self.name = name # Assign the provided name to the instance variable 'name'
        self.roll_no = roll_no # Assign the provided roll number to the instance variable 'roll_no'
        self.hostel_status = hostel_status # Assign the provided hostel status to the instance variable 'hostel_status'
        self.fees_paid = 0 # Initialize the fees_paid attribute to 0 for a new student

    def fee_update(self, amount): # Method to update the fee payment for a student
        """Updates the student's fee payment.

        Args:
            amount: The amount of fee paid.
        """ # Docstring explaining the fee_update method and its argument
        if amount > 0: # Check if the payment amount is positive
            self.fees_paid += amount # Add the paid amount to the total fees paid
            print(f"Fee of {amount} updated for {self.name}. Total fees paid: {self.fees_paid}") # Print a confirmation message
        else: # If the amount is not positive
            print("Invalid amount. Fee amount must be positive.") # Print an error message

    def display_details(self): # Method to display the details of a student
        """Displays the student's details.""" # Docstring explaining the display_details method
        print("\nStudent Details:") # Print a header for the student details
        print(f"Name: {self.name}") # Print the student's name
        print(f"Roll No.: {self.roll_no}") # Print the student's roll number
        print(f"Hostel Status: {self.hostel_status}") # Print the student's hostel status
        print(f"Fees Paid: {self.fees_paid}") # Print the total fees paid by the student

```




```
# Example usage:
student1 = sru_student("Alice", "SRU123", "resident") # Create the first student object
student2 = sru_student("Bob", "SRU456", "day scholar") # Create the second student object

student1.display_details() # Display details for the first student
student2.display_details() # Display details for the second student

student1.fee_update(5000) # Update fee payment for the first student
student2.fee_update(2000) # Update fee payment for the second student
student1.fee_update(3000) # Another fee update for the first student

student1.display_details() # Display updated details for the first student
student2.display_details() # Display updated details for the second student
```



```
Student Details:
Name: Alice
Roll No.: SRU123
Hostel Status: resident
Fees Paid: 0

Student Details:
Name: Bob
Roll No.: SRU456
Hostel Status: day scholar
Fees Paid: 0
Fee of 5000 updated for Alice. Total fees paid: 5000
Fee of 2000 updated for Bob. Total fees paid: 2000
Fee of 3000 updated for Alice. Total fees paid: 8000

Student Details:
Name: Alice
Roll No.: SRU123
Hostel Status: resident
Fees Paid: 8000

Student Details:
Name: Bob
Roll No.: SRU456
Hostel Status: day scholar
Fees Paid: 2000
```

COMPARISON:

- The first code which was given manually uses inline comments (#) for line-by-line explanations, focusing on what the code does.
- Second code uses both inline comments and structured docstrings which was given by the AI. Docstrings explain the purpose of classes/methods, their parameters, and provide examples (NumPy style).
- Inline comments are good for specific lines, while docstrings offer detailed, structured documentation. My code combines both for a comprehensive approach.

OBSERVATION:

- The provided Python code defines an `sru_student` class to manage student information. It includes a constructor to initialize student objects with name, roll number, and hostel status, and sets initial fees paid to zero. The `fee_update` method allows adding fee payments to a student's record. The `display_details` method prints the student's information, including their name, roll number, hostel status, and total fees paid. The example usage demonstrates creating student objects, updating fees, and displaying details.

Task Description#3


- Write a Python script with 3–4 functions (e.g., calculator: add, subtract, multiply, divide).
- Incorporate manual **docstring** in code with NumPy Style
- Use AI assistance to generate a module-level docstring + individual function docstrings.
- Compare the AI-generated docstring with your manually written one.


Expected Output#3: Students learn structured documentation for multi-function scripts

PROMPT:

- Write a Python script with functions for add, subtract, multiply, and divide, each using NumPy-style docstrings, including a module-level docstring.

CODE:

```
 def add(a, b):  
    return a + b  
  
def subtract(a, b):  
    return a - b  
  
def multiply(a, b):  
    return a * b  
  
def divide(a, b):  
    if b == 0:  
        raise ZeroDivisionError("division by zero")  
    return a / b  
  
# Example usage:  
print(f"Addition: {add(10, 5)}")  
print(f"Subtraction: {subtract(10, 5)}")  
print(f"Multiplication: {multiply(10, 5)}")  
print(f"Division: {divide(10, 5)}")  
  
# Example of ZeroDivisionError  
try:  
    print(f"Division by zero: {divide(10, 0)}")  
except ZeroDivisionError as e:  
    print(e)
```

```
 Addition: 15  
Subtraction: 5  
Multiplication: 50  
Division: 2.0  
division by zero
```

	<p>OBSERVATION:</p> <ul style="list-style-type: none">• The code defines four Python functions: add, subtract, multiply, and divide. These functions are designed to perform basic arithmetic operations using NumPy's corresponding functions (np.add, np.subtract, np.multiply, and np.divide). This approach allows the functions to work seamlessly with both scalar values and NumPy arrays, providing element-wise operations similar to how NumPy handles arithmetic. <p>Push documentation whole workspace as .md file in GitHub Repository</p> <p>Note: Report should be submitted a word document for all tasks in a single document with prompts, comments & code explanation, and output and if required, screenshots</p>	
--	--	--