

PROGRAM : B.TECH/CSE
SPECIALIZATION : AIML
COURSE TITLE : AI ASSISTANT CODING
COURSE CODE : 24CS101PC214
SEMESTER : 3RD
NAME OF THE STUDENT : KARNAKANTI GODADEVI
ENROLLMENT NO : 2403A52003
BATCH NO : 01

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 that returns the sum of even and odd numbers in a given list.
generate the docstring function automatically.

Code&Output:

The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'google colab - Yahoo In...', 'Welcome To Colab - Col...', 'Untitled7.ipynb - Colab', 'assignment 9.3', and 'Docstring generation co...'. The address bar shows the URL 'colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=4f1987df'. The notebook is titled 'Untitled7.ipynb' and has a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu bar is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The main code cell contains a Python function definition with a docstring and example usage:

```
[6] 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  
    # Example usage of the sum_even_odd function  
numbers_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
even_sum, odd_sum = sum_even_odd(numbers_list)  
  
print(f"The list is: {numbers_list}")  
print(f"Sum of even numbers: {even_sum}")  
print(f"Sum of odd numbers: {odd_sum}")
```

The output of the code cell is displayed below the code:

```
The list is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The bottom of the notebook shows a 'Variables' panel, a 'Terminal' panel, and a status bar indicating '2:09 PM', 'Python 3', and '14:37 24-09-2025'.

The screenshot shows the same Google Colab notebook interface as the first image. The code cell contains the same Python function definition with a docstring and example usage:

```
[6] 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  
    # Example usage of the sum_even_odd function  
numbers_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
even_sum, odd_sum = sum_even_odd(numbers_list)  
  
print(f"The list is: {numbers_list}")  
print(f"Sum of even numbers: {even_sum}")  
print(f"Sum of odd numbers: {odd_sum}")
```

The output of the code cell is displayed below the code:

```
The list is: [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
Sum of even numbers: 30  
Sum of odd numbers: 25
```

The bottom of the notebook shows a 'Variables' panel, a 'Terminal' panel, and a status bar indicating '2:09 PM', 'Python 3', and '14:37 24-09-2025'.

Observations:

- This function efficiently separates and sums even and odd numbers.
- It iterates through the list once, making it performant.
- The use of a docstring clearly explains the function's purpose and parameters.

Task Description#2 Automatic Inline Comments

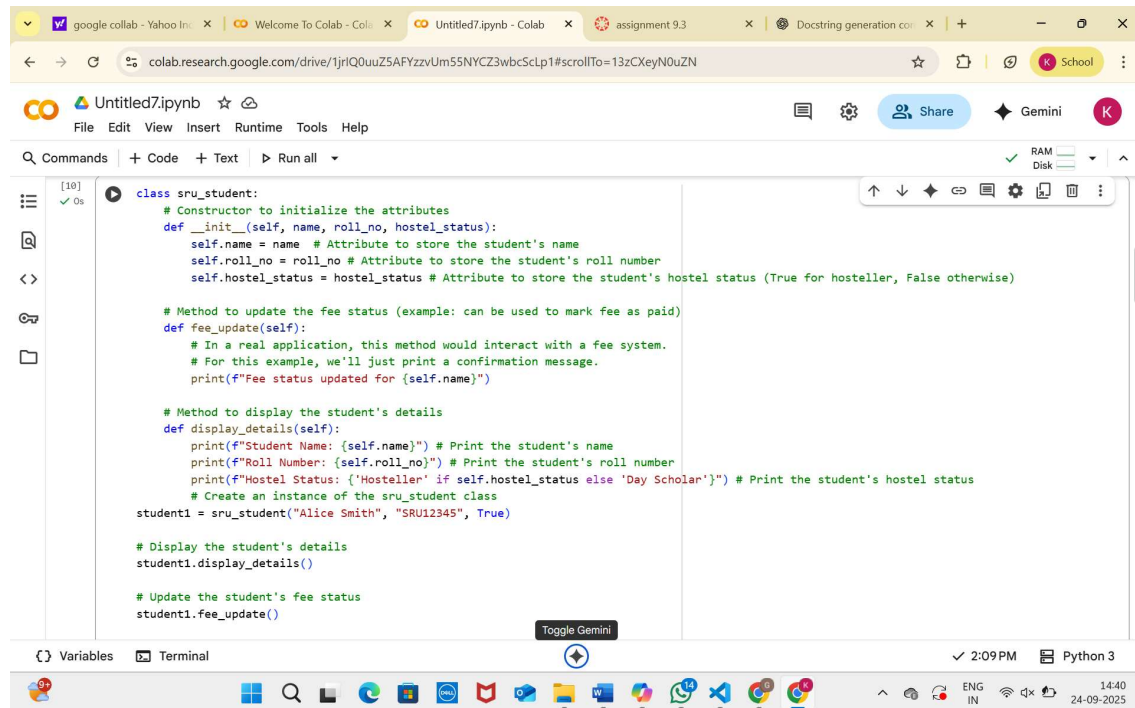
- Write python program for sru_student class with attributes like name, roll no., hostel_status and fee_update method and display_details method.
- Write comments manually for each line/code block
- Ask an AI tool to add inline comments explaining each line/step.
- Compare the AI-generated comments with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

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

Prompt:

Write a Python class named sru_student with attributes (name, roll_no, hostel_status) and methods fee_update and display_details. automatically add inline comments explaining each line of code.

Code&Output:



```
[10] ✓ 0s
class sru_student:
    # Constructor to initialize the attributes
    def __init__(self, name, roll_no, hostel_status):
        self.name = name # Attribute to store the student's name
        self.roll_no = roll_no # Attribute to store the student's roll number
        self.hostel_status = hostel_status # Attribute to store the student's hostel status (True for hosteller, False otherwise)

    # Method to update the fee status (example: can be used to mark fee as paid)
    def fee_update(self):
        # In a real application, this method would interact with a fee system.
        # For this example, we'll just print a confirmation message.
        print(f"Fee status updated for {self.name}")

    # Method to display the student's details
    def display_details(self):
        print(f"Student Name: {self.name}") # Print the student's name
        print(f"Roll Number: {self.roll_no}") # Print the student's roll number
        print(f"Hostel Status: {'Hosteller' if self.hostel_status else 'Day Scholar'}") # Print the student's hostel status
        # Create an instance of the sru_student class
        student1 = sru_student("Alice Smith", "SRU12345", True)

    # Display the student's details
    student1.display_details()

    # Update the student's fee status
    student1.fee_update()
```

The screenshot shows a Google Colab notebook titled 'Untitled7.ipynb'. The code defines a `Student` class with methods `display_details` and `fee_update`. It creates two instances, `student1` (Alice Smith) and `student2` (Bob Johnson), and prints their details and updated fee status. The output shows the student details and the fee status update for Alice Smith.

```
print(f"Fee status updated for {self.name}")

# Method to display the student's details
def display_details(self):
    print(f"Student Name: {self.name}") # Print the student's name
    print(f"Roll Number: {self.roll_no}") # Print the student's roll number
    print(f"Hostel Status: {'Hosteller' if self.hostel_status else 'Day Scholar'}") # Print the student's hostel status
    # Create an instance of the sru_student class
    student1 = sru_student("Alice Smith", "SRU12345", True)

    # Display the student's details
    student1.display_details()

    # Update the student's fee status
    student1.fee_update()

    # Create another instance and display details
    student2 = sru_student("Bob Johnson", "SRU67890", False)
    student2.display_details()
```

Student Name: Alice Smith
Roll Number: SRU12345
Hostel Status: Hosteller
Fee status updated for Alice Smith
Student Name: Bob Johnson
Roll Number: SRU67890
Hostel Status: Day Scholar

Observations:

- Classes are useful for grouping data (attributes) and functionality (methods).
- The constructor (`__init__`) is used to set up the initial state of an object.
- Methods allow objects to perform actions or display information.

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 3–4 functions (e.g., add, subtract, multiply, divide) generate: A module-level docstring. Function-level docstrings. Expected Outcome: Students observe how AI creates structured documentation (NumPy style or similar) for a full script.

Code&Output:

google collab - Yahoo In... x Welcome To Colab - Col... x Untitled7.ipynb - Colab x assignment 9.3 x Docstring generation co... x + -

colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB

Untitled7.ipynb File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

```
"""
This module provides basic arithmetic functions.

Functions:
    add(a, b): Returns the sum of two numbers.
    subtract(a, b): Returns the difference between two numbers.
    multiply(a, b): Returns the product of two numbers.
    divide(a, b): Returns the division of two numbers.
"""

def add(a, b):
    """
    Adds two numbers.

    Parameters
    -----
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -----
    float
        The sum of the two numbers.
    """
    return a + b
```

Variables Terminal 2:09 PM Python 3

google collab - Yahoo In... x Welcome To Colab - Col... x Untitled7.ipynb - Colab x assignment 9.3 x Docstring generation co... x + -

colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB

Untitled7.ipynb File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

RAM Disk

```
"""
    return a + b

def subtract(a, b):
    """
    Subtracts the second number from the first.

    Parameters
    -----
    a : float
        The first number.
    b : float
        The second number.

    Returns
    -----
    float
        The difference between the two numbers.
    """
    return a - b

def multiply(a, b):
    """
    Multiplies two numbers.

    Parameters
    -----
    a : float
```

Variables Terminal 2:09 PM Python 3

google collab - Yahoo In... x Welcome To Colab - Col... x Untitled7.ipynb - Colab x assignment 9.3 x Docstring generation co... x + -

colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB

Untitled7.ipynb File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

[13] ✓ 0s

```
a : float
    The first number.
b : float
    The second number.

Returns
-----
float
    The product of the two numbers.
"""
return a * b

def divide(a, b):
    """
    Divides the first number by the second.

    Parameters
    -----
    a : float
        The numerator.
    b : float
        The denominator.

    Returns
    -----
    float
        The result of the division.
    """
```

{ } Variables Terminal 2:09 PM Python 3

google collab - Yahoo In... x Welcome To Colab - Col... x Untitled7.ipynb - Colab x assignment 9.3 x Docstring generation co... x + -

colab.research.google.com/drive/1jrlQ0uuZ5AFYzzvUm55NYCZ3wbcScLp1#scrollTo=MR3SsvBw3cWB

Untitled7.ipynb File Edit View Insert Runtime Tools Help

Q Commands + Code + Text ▶ Run all

[13] ✓ 0s

```
float
    The result of the division.

Raises
-----
ZeroDivisionError
    If the denominator is zero.
"""
if b == 0:
    raise ZeroDivisionError("Division by zero is not allowed.")
return a / b
# Example usage of the arithmetic functions
num1 = 10
num2 = 5

sum_result = add(num1, num2)
print(f"The sum of {num1} and {num2} is: {sum_result}")

difference_result = subtract(num1, num2)
print(f"The difference between {num1} and {num2} is: {difference_result}")

product_result = multiply(num1, num2)
print(f"The product of {num1} and {num2} is: {product_result}")

# Example of division, including handling potential ZeroDivisionError
try:
    division_result = divide(num1, num2)
    print(f"The division of {num1} by {num2} is: {division_result}")
```

{ } Variables Terminal 2:09 PM Python 3

The screenshot shows a Google Colab notebook titled 'Untitled7.ipynb'. The code in the cell includes a multiplication function, a division function with a try-except block for ZeroDivisionError, and a series of print statements. The output shows the results of these operations: the sum of 10 and 5 is 15, the difference between 10 and 5 is 5, the product of 10 and 5 is 50, and the division of 10 by 5 is 2.0. The interface includes a menu bar with File, Edit, View, Insert, Runtime, Tools, and Help. The bottom status bar shows the time as 2:09 PM and the Python version as Python 3.

```
[13] ✓ 0s
product_result = multiply(num1, num2)
print(f"The product of {num1} and {num2} is: {product_result}")

# Example of division, including handling potential ZeroDivisionError
try:
    division_result = divide(num1, num2)
    print(f"The division of {num1} by {num2} is: {division_result}")

    # Example of division by zero to show the error handling
    # division_by_zero_result = divide(num1, 0)
    # print(f"Division by zero example: {division_by_zero_result}")

except ZeroDivisionError as e:
    print(f"Error: {e}")

The sum of 10 and 5 is: 15
The difference between 10 and 5 is: 5
The product of 10 and 5 is: 50
The division of 10 by 5 is: 2.0
```

Observations:

- AI tools can effectively generate structured docstrings for Python code.
- Module-level docstrings provide a high-level overview of the script's purpose.
- Function-level docstrings detail the parameters, return values, and potential errors of each function.