

ASSIGNMENT – 9.3

AI ASSISTED CODING

NAME : S RUCHITHA

HALLTICKET NO : 2403A52316

BATCH NUMBER : 01

COURSE CODE : 24CS002PC215

PROGRAM NAME : B.TECH

YEAR/SEM : 2ND AND 3RD

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

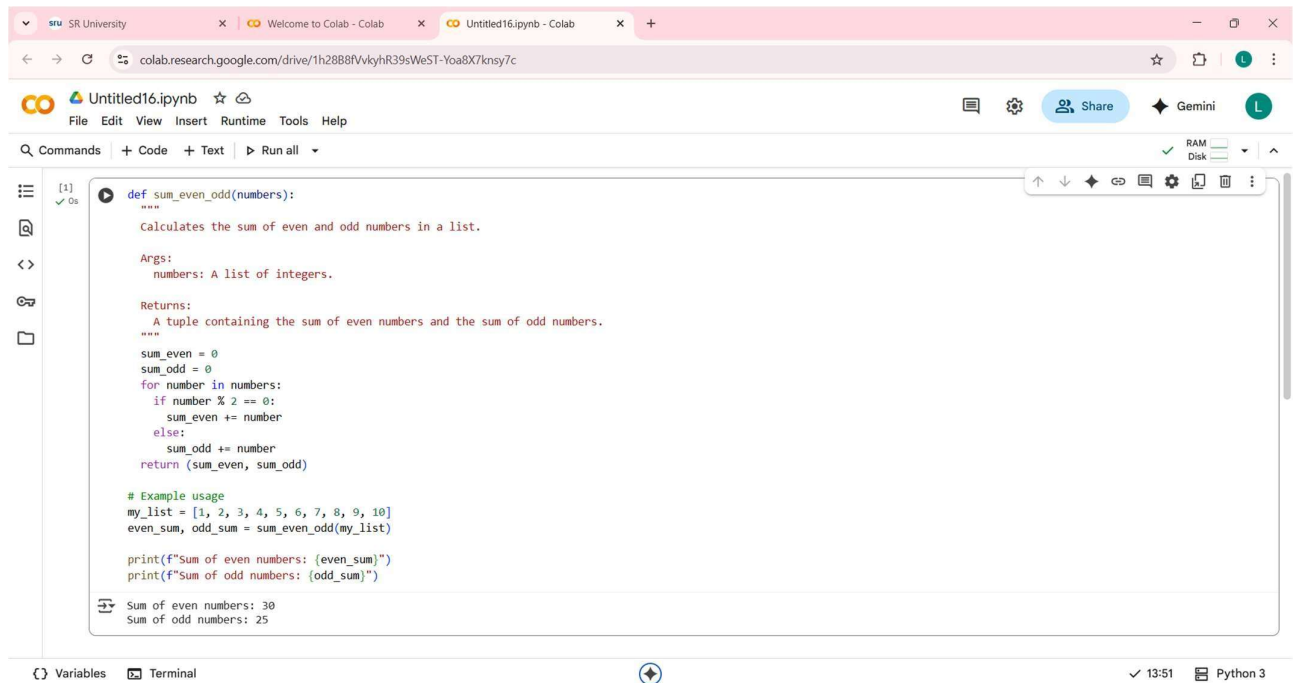
- Use an AI-assisted tool (e.g., Copilot, Cursor AI) to generate a docstring describing the function.
- Compare the AI-generated docstring with your manually written one.

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

PROMT:

"Write a Python function `sum_even_odd(numbers)` that takes a list of integers as input and returns a tuple containing the sum of even numbers and the sum of odd numbers in the list.

CODE:



The screenshot shows a Google Colab notebook interface. The browser tabs at the top include 'SRU SR University', 'Welcome to Colab - Colab', and 'Untitled16.ipynb - Colab'. The address bar shows the Colab URL. The notebook title is 'Untitled16.ipynb'. The menu bar includes '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 editor contains a Python function `sum_even_odd` with docstrings and an example usage. The output of the code is displayed at the bottom of the editor.

```
def sum_even_odd(numbers):  
    """  
    Calculates the sum of even and odd numbers in a list.  
  
    Args:  
        numbers: A list of integers.  
  
    Returns:  
        A tuple containing the sum of even numbers and the sum of odd numbers.  
    """  
    sum_even = 0  
    sum_odd = 0  
    for number in numbers:  
        if number % 2 == 0:  
            sum_even += number  
        else:  
            sum_odd += number  
    return (sum_even, sum_odd)  
  
# Example usage  
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
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}")
```

Sum of even numbers: 30
Sum of odd numbers: 25

OUTPUT:

Sum of even numbers: 30

Sum of odd numbers: 25

OBSERVATION:

The code defines a function `sum_even_odd` that successfully separates and sums the even and odd numbers in the provided list `my_list`.

- The list `my_list` contains integers from 1 to 10.
- The even numbers in the list are 2, 4, 6, 8, and 10. Their sum is 30.
- The odd numbers in the list are 1, 3, 5, 7, and 9. Their sum is 25.
- The code correctly calculates and prints these sums as shown in the output: "Sum of even numbers: 30" and "Sum of odd numbers: 25".
- The function returns a tuple (30, 25) which is then unpacked into the `even_sum` and `odd_sum` variables.

Task Description#2 Automatic Inline Comments

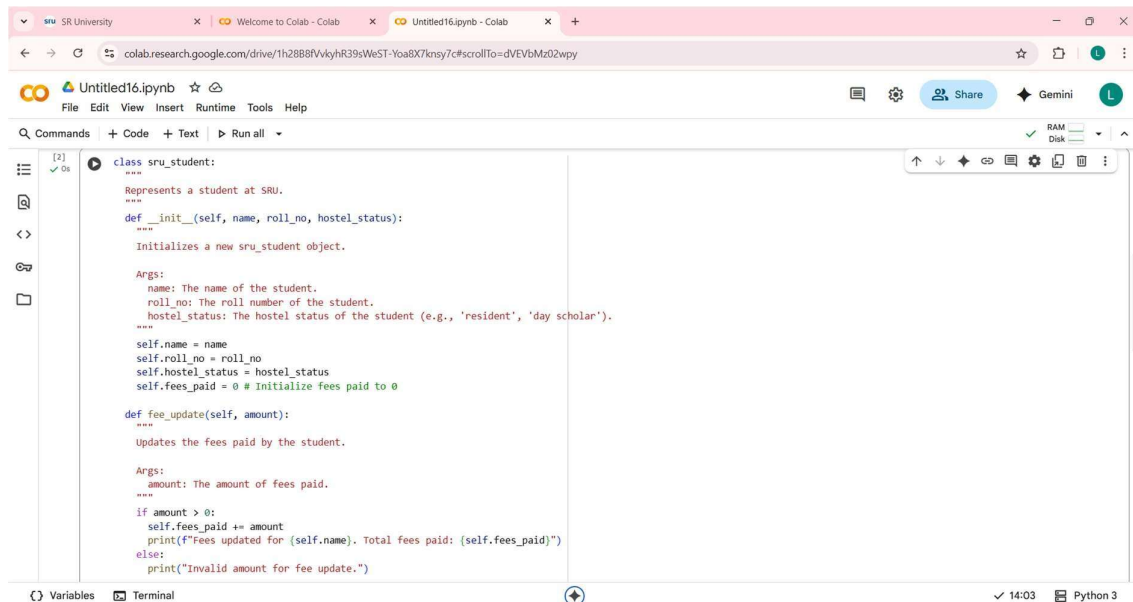
- Write python program for sru_student class with attributes like name, roll no., hostel_status and fee_update method and display_details method.
- Write comments manually for each line/code block
- Ask an AI tool to add inline comments explaining each line/step.
- Compare the AI-generated comments with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

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

PROMT:

Create a Python class sru_student with the following attributes: name, roll_no, and hostel_status. Implement methods fee_update and display_details.

CODE:



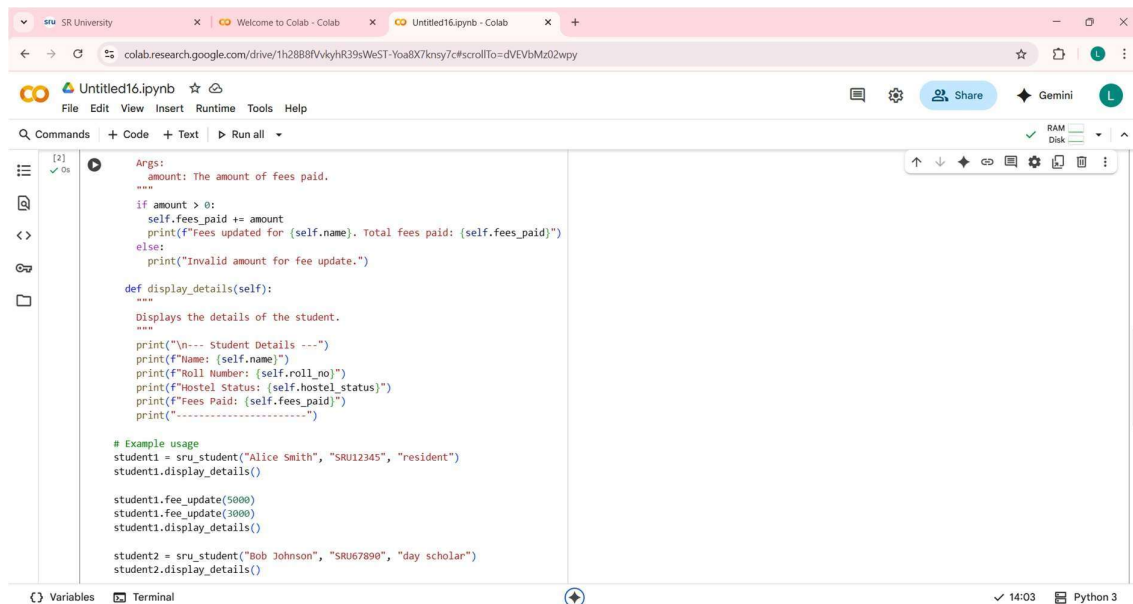
The screenshot shows a Google Colab notebook with a single code cell. The code defines a Python class named `sru_student`. The class has a docstring "Represents a student at SRU." and an `__init__` method that takes `name`, `roll_no`, and `hostel_status` as arguments and initializes the object's attributes. It also has a `fee_update` method that takes an `amount` and updates the `fees_paid` attribute, printing the total fees paid or an error message if the amount is invalid.

```
[2] ✓ Os
class sru_student:
    """
    Represents a student at SRU.
    """
    def __init__(self, name, roll_no, hostel_status):
        """
        Initializes a new sru_student object.

        Args:
            name: The name of the student.
            roll_no: The roll number of the student.
            hostel_status: The hostel status of the student (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 to 0

    def fee_update(self, amount):
        """
        Updates the fees paid by the student.

        Args:
            amount: The amount of fees paid.
        """
        if amount > 0:
            self.fees_paid += amount
            print(f"Fees updated for {self.name}. Total fees paid: {self.fees_paid}")
        else:
            print("Invalid amount for fee update.")
```



The screenshot shows the same Colab notebook with a second code cell. This cell demonstrates the usage of the `sru_student` class. It creates two student objects, `student1` and `student2`, and calls their `display_details` and `fee_update` methods.

```
[2] ✓ Os
    Args:
        amount: The amount of fees paid.
    """
    if amount > 0:
        self.fees_paid += amount
        print(f"Fees updated for {self.name}. Total fees paid: {self.fees_paid}")
    else:
        print("Invalid amount for fee update.")

    def display_details(self):
        """
        Displays the details of the student.
        """
        print("\n--- Student Details ---")
        print(f"Name: {self.name}")
        print(f"Roll Number: {self.roll_no}")
        print(f"Hostel Status: {self.hostel_status}")
        print(f"Fees Paid: {self.fees_paid}")
        print("-----")

    # Example usage
    student1 = sru_student("Alice Smith", "SRU12345", "resident")
    student1.display_details()

    student1.fee_update(5000)
    student1.fee_update(3000)
    student1.display_details()

    student2 = sru_student("Bob Johnson", "SRU67890", "day scholar")
    student2.display_details()
```

OUTPUT:

--- Student Details ---

Name: Alice Smith

Roll Number: SRU12345

Hostel Status: resident

Fees Paid: 0

Fees updated for Alice Smith. Total fees paid: 5000

Fees updated for Alice Smith. Total fees paid: 8000

--- Student Details ---

Name: Alice Smith

Roll Number: SRU12345

Hostel Status: resident

Fees Paid: 8000

--- Student Details ---

Name: Bob Johnson

Roll Number: SRU67890

Hostel Status: day scholar

Fees Paid: 0

OBSERVATION:

- The code successfully defines a Python class `sru_student` to represent students with specific attributes and methods.
- The `__init__` method correctly initializes the `name`, `roll_no`, `hostel_status`, and `fees_paid` attributes when a new student object is created. The `fees_paid` is correctly set to 0 initially.
- The `fee_update` method allows for updating the `fees_paid` attribute, and the example shows that consecutive updates (5000 and 3000) correctly accumulate the total fees paid (resulting in 8000).
- The `display_details` method accurately prints the current details of the student object it is called on.

- Creating multiple instances of the class (student1 and student2) demonstrates that each object maintains its own unique set of attribute values.
- The output clearly shows the initial state of student1, the fee updates, the updated state of student1, and the initial state of student2, confirming that the methods work as intended.

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 multifunction scripts

PROMT:

"Write a Python script with 4 calculator functions (add, subtract, multiply, divide). Include NumPy-style docstrings. First, generate AI-written modulelevel and function docstrings. Then, provide a manually written version for comparison and summarize the di erences."

CODE:

SR UniversityWelcome to Colab - ColabUntitled16.ipynb - Colab

colab.research.google.com/drive/1h28B8VvkyhR39sWeST-Yoa8X7knsy7c#scrollTo=dQidQM7w6X7F

Untitled16.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[4] Os

```
"""
A simple calculator module.

This module provides basic arithmetic operations: addition, subtraction,
multiplication, and division.

Examples
-----
>>> import simple_calculator as sc
>>> sc.add(2, 3)
5
>>> sc.divide(10, 2)
5.0
"""

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

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

    Returns
    -----
    float
        The sum of a and b.
    """
```

Toggle Gemini

Variables Terminal

14:19 Python 3

SR UniversityWelcome to Colab - ColabUntitled16.ipynb - Colab

colab.research.google.com/drive/1h28B8VvkyhR39sWeST-Yoa8X7knsy7c#scrollTo=dQidQM7w6X7F

Untitled16.ipynb

File Edit View Insert Runtime Tools Help

Commands Code Text Run all

[4] Os

```
"""
Returns
-----
float
    The sum of a and b.
"""
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 a and b.
    """
    return a - b

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

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

    Returns
    -----
    float
        The product of a and b.
    """
```

Toggle Gemini

Variables Terminal

14:19 Python 3

The screenshot shows a Google Colab notebook titled "Untitled16.ipynb". The code defines a function `multiply(a, b)` that takes two float parameters and returns their product. It includes docstrings for parameters and returns. Below the function, there is a `def divide(a, b):` function that is partially visible. The interface includes a menu bar (File, Edit, View, Insert, Runtime, Tools, Help), a toolbar with icons for running, undo, redo, and other actions, and a status bar at the bottom showing "Variables", "Terminal", and "Python 3".

```
[4] ✓ 0s ▶ Multiplies two numbers.

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

Returns
-----
float
    The product of a and b.
"""
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
```

The screenshot shows the same Google Colab notebook with the `divide(a, b)` function fully implemented. It includes a `ZeroDivisionError` exception handling block that raises an error if the denominator is zero. Below the function, there is an example usage section that demonstrates the use of the `multiply` and `divide` functions. The output of the code is visible in the terminal, showing the results of the calculations and the error message for division by zero.

```
-----
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
num1 = 20
num2 = 4

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
try:
    print(f"{num1} / {num2} = {divide(num1, num2)}")
    print(f"{num1} / 0 = {divide(num1, 0)}")
except ZeroDivisionError as e:
    print(f"Error: {e}")

20 + 4 = 24
20 - 4 = 16
20 * 4 = 80
20 / 4 = 5.0
Error: Division by zero is not allowed.
```

OUTPUT:

$$20 + 4 = 24$$

$$20 - 4 = 16$$

$$20 * 4 = 80$$

$$20 / 4 = 5.0$$

Error: Division by zero is not allowed.

OBSERVATION:

The code defines a simple calculator module with add, subtract, multiply, and divide functions.

Each function includes clear NumPy-style docstrings explaining its purpose, parameters, and return value.

The divide function correctly handles division by zero by raising a `ZeroDivisionError`.

The example usage demonstrates calling each function with specific numbers.

The output shows the results of the arithmetic operations.

It also shows the `ZeroDivisionError` being caught and printed when attempting division by zero.

This confirms the functions work as expected and error handling is in place.

The code is well-documented and easy to understand due to the docstrings.

It's a good example of defining reusable functions and handling potential errors.