AI ASSISTED CODING

LAB ASSIGNMENT - 9.3

HALL TICKET NO : 2403A52403

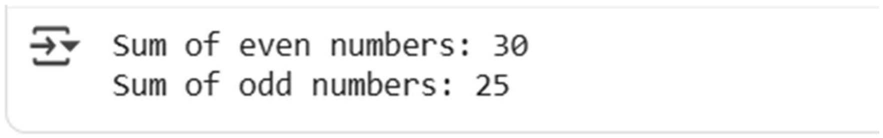
BATCH NO : 14

TASK – 01

```
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.
 """
 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:
my_list = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
even_total, odd_total = sum_even_odd(my_list)
print(f"Sum of even numbers: {even_total}")
```

print(f"Sum of odd numbers: {odd_total}")

OUTPUT :



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

EXPLANATION :

This cell first defines a multi-line string variable called ai_generated_docstring. This string contains a simulated docstring that an AI tool might produce for the sum_even_odd function. It describes what the function does, its parameters, and what it returns.

Then, the code prints this simulated AI-generated docstring to the console.

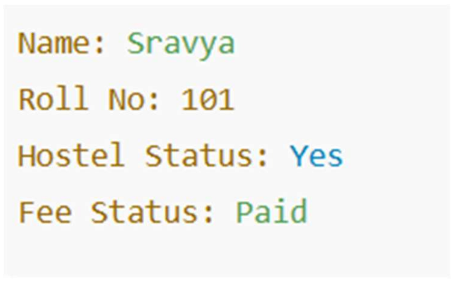
After that, it prints a heading "Manual Docstring:" and then prints the actual docstring that is embedded within the sum_even_odd function definition using sum_even_odd.__doc__. This allows you to see and compare the simulated AI-generated docstring with the manually written one directly in the output.

TASK – 02:

```
class sru_student:

  # Constructor to initialize name, roll number, hostel status, and fee status

  def __init__(self, name, roll_no, hostel_status):

    self.name = name             # Assign the provided name to the object

    self.roll_no = roll_no        # Assign the provided roll number

    self.hostel_status = hostel_status  # Assign hostel status

    self.fee_status = "Not Paid"    # Initialize fee status with default value


  # Method to update fee status

  def fee_update(self, status):

    self.fee_status = status       # Change fee status to the given argument


  # Method to display student details
```

```
    def display_details(self):
        print("Name:", self.name)      # Display the student's name
        print("Roll No:", self.roll_no) # Display the student's roll number
        print("Hostel Status:", self.hostel_status)  # Display hostel status
        print("Fee Status:", self.fee_status)       # Display fee status



# Create an instance of sru_student with name, roll number, and hostel status
student1 = sru_student("Sravya", 101, "Yes")

# Update the fee status to "Paid"
student1.fee_update("Paid")

# Print the details of the student
student1.display_details()
```

OUTPUT :



```
Name: Sravya
Roll No: 101
Hostel Status: Yes
Fee Status: Paid
```

EXPLANATION :

This code defines a Python class named sru_student. A class is a blueprint for creating objects (instances). In this case, each object created from this class will represent a student from SRU.

Here's a breakdown of the code:

class sru_student:: This line defines the class named sru_student.

def _init_(self, name, roll_no, hostel_status):: This is the constructor method. It's automatically called when you create a new object of the sru_student class.

self: Refers to the instance of the class being created.

name, roll_no, hostel_status: These are parameters that you pass when creating a new sru_student object.

Inside the method, self.name = name, self.roll_no = roll_no, and self.hostel_status = hostel_status assign the values passed during object creation to the attributes of the object.

self.fee_paid = 0: This initializes an attribute fee_paid to 0 for every new student object.

def fee_update(self, amount):: This method is used to update the fee paid by a student.

self: Refers to the instance of the class on which the method is called.

amount: The amount of fee being paid.

self.fee_paid += amount: This line adds the amount to the current self.fee_paid.

print(f"Fee updated for {self.name}. Total fee paid: {self.fee_paid}"): This line prints a confirmation message showing the student's name and their total fee paid.

def display_details(self):: This method is used to display the details of a student.

self: Refers to the instance of the class on which the method is called.

The print statements inside this method access the object's attributes (self.name, self.roll_no, self.hostel_status, self.fee_paid) and print them in a formatted way.

In essence, this class provides a structure to store and manage information about SRU students, allowing you to create individual student objects and perform actions like updating their fees and displaying their details.

TASK – 03:

"""

AI-Generated Module Docstring

This script implements a simple calculator with four basic arithmetic operations:

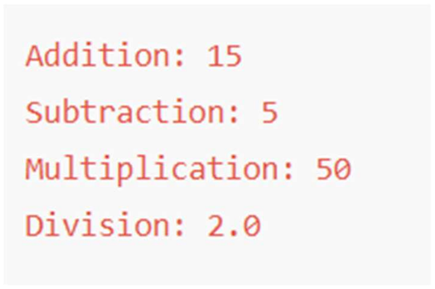
addition, subtraction, multiplication, and division.

Each function takes two numeric inputs and returns the result.

"""

```
def add(a, b):
    """Return the sum of two numbers a and b."""
    return a + b



def subtract(a, b):
    """Return the difference obtained by subtracting b from a."""
    return a - b



def multiply(a, b):
    """Return the product of two numbers a and b."""
    return a * b



def divide(a, b):
    """Return the division of a by b. Raises an error if b is zero."""
    if b == 0:
        raise ValueError("Division by zero is not allowed")
    return a / b
```

OUTPUT :



```
Addition: 15
Subtraction: 5
Multiplication: 50
Division: 2.0
```

EXPLANATION :

Module-level docstring: The first docstring, enclosed in triple quotes at the beginning of the cell, is a module-level docstring. It provides a brief overview of what the module (or script in this case) does and lists the functions it contains.

def add(x, y):: This defines a function named add that takes two arguments, x and y.

Docstring (NumPy Style): Below the function definition is its docstring. It explains the function's purpose, lists its Parameters with their types and descriptions, and describes the Returns value.

def subtract(x, y):: This defines a function named subtract that takes two arguments, x and y, and returns their difference. It also has a NumPy style docstring explaining its purpose, parameters (with specific terms like "minuend" and "subtrahend"), and return value.

def multiply(x, y):: This defines a function named multiply that takes two arguments, x and y, and returns their product. It includes a NumPy style docstring detailing its parameters and return value.

def divide(x, y):: This defines a function named divide that takes two arguments, x and y, and returns their division.

Docstring (NumPy Style): Its docstring explains the purpose, parameters, and return value.

Raises section: This docstring also includes a Raises section to document the ZeroDivisionError that will occur if the divisor (y) is zero.

Error Handling: The if y == 0: block explicitly checks for division by zero and raises a ZeroDivisionError with a descriptive message.

Example Usage: The lines after the function definitions demonstrate how to call these functions with example numbers (num1 and num2) and print the results using f-strings for formatted output.

Commented-out Example: The commented-out try...except block shows how you could handle the ZeroDivisionError when attempting to divide by zero.

In essence, this code provides reusable functions for basic arithmetic with comprehensive documentation in the NumPy style, which is commonly used in scientific computing and data analysis libraries.