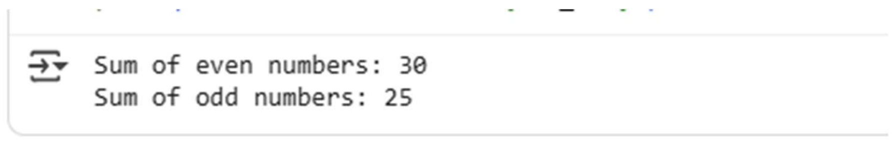
AI ASSITED CODING
LAB 9.3
HALL TICKT NO : 2403A52394
BATCH NO : 14

TASK 1

```
def sum_even_odd(numbers):
 """Calculates the sum of even and odd numbers in a list.

 Args:
  numbers: A list of numbers (integers or floats).

 Returns:
  A tuple containing two elements: 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_sum, odd_sum = sum_even_odd(my_list)

print(f"Sum of even numbers: {even_sum}")

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

OUT PUT :



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

EXPLANATION :

The code defines a Python function called sum_even_odd that takes a list of numbers as input.

Here's a breakdown of the code:

1. **def sum_even_odd(numbers):**: This line defines the function named sum_even_odd and indicates that it accepts one argument, numbers.
2. **Docstring**: The multi-line string within triple quotes ("""...""") is a docstring. It explains what the function does, its arguments (Args), and what it returns (Returns). This specific docstring follows the Google style.
3. **even_sum = 0 and odd_sum = 0**: These lines initialize two variables, even_sum and odd_sum, to zero. These variables will store the running total of even and odd numbers, respectively.
4. **for number in numbers:**: This is a for loop that iterates through each element in the input numbers list. In each iteration, the current element is assigned to the variable number.
5. **if number % 2 == 0:**: This is a conditional statement that checks if the current number is even. The modulo operator (%) returns the remainder of a division. If a number divided by 2 has a remainder of 0, it's an even number.
6. **even_sum += number**: If the condition in the if statement is true (the number is even), this line adds the current number to the even_sum.
7. **else:**: This indicates what to do if the condition in the if statement is false (the number is not even, meaning it's odd).

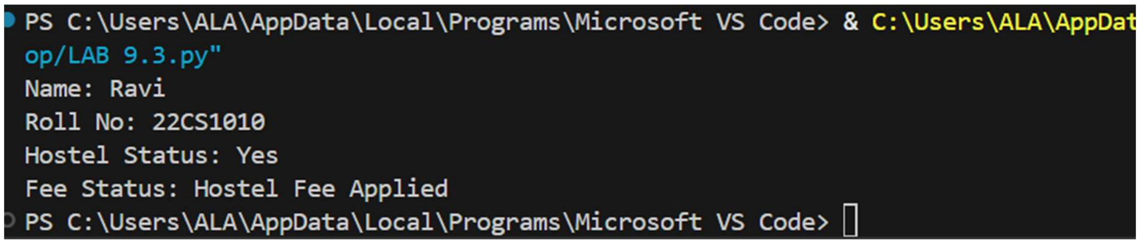
8. **odd_sum += number**: If the number is odd, this line adds the current number to the odd_sum.
9. **return even_sum, odd_sum**: After the loop finishes iterating through all the numbers in the list, the function returns a tuple containing the final even_sum and odd_sum.
10. **Example usage**: The lines after the function definition show how to call the function with a sample list my_list and print the results.

In summary, the function efficiently separates and sums the even and odd numbers within any given list.

TASK 2

```
class sru_student:
  # Initialize the student object with name, roll number, and hostel status
  def __init__(self, name, roll_no, hostel_status):
    self.name = name              # Assign the name to the student object
    self.roll_no = roll_no        # Assign the roll number to the student
    self.hostel_status = hostel_status  # Assign hostel status ('Yes' or 'No')
    self.fee_status = None        # Initialize fee status as None (to be updated later)

  # Update the fee status depending on whether the student is in a hostel
  def fee_update(self):
    if self.hostel_status.lower() == 'yes':   # If the student stays in a hostel (case-
insensitive check)
      self.fee_status = 'Hostel Fee Applied'  # Set fee status to indicate hostel fees
    else:
      self.fee_status = 'Day Scholar Fee Applied'  # Set fee status for non-hostel
students

  # Print all the details of the student
  def display_details(self):
    print(f"Name: {self.name}")          # Display the student's name
    print(f"Roll No: {self.roll_no}")      # Display the student's roll number
    print(f"Hostel Status: {self.hostel_status}")  # Display hostel status
    print(f"Fee Status: {self.fee_status}")   # Display the fee status
    # Create a student object
student1 = sru_student("Ravi", "22CS1010", "Yes")
student1.fee_update()      # Update the fee based on hostel status
student1.display_details()   # Display all details
```

OUT PUT:



```
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code> & C:\Users\ALA\AppDat
op/LAB 9.3.py"
Name: Ravi
Roll No: 22CS1010
Hostel Status: Yes
Fee Status: Hostel Fee Applied
PS C:\Users\ALA\AppData\Local\Programs\Microsoft VS Code>
```

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.

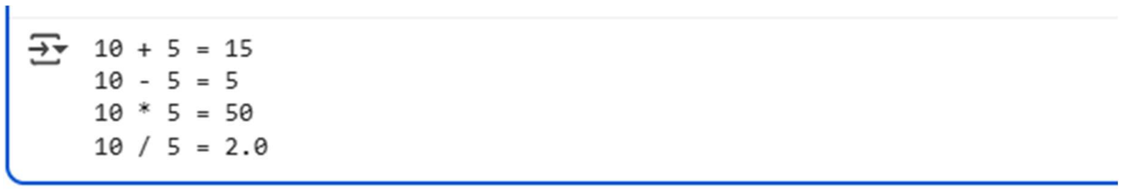
TASK 3

```
Functions:
  add(x, y): Adds two numbers.
  subtract(x, y): Subtracts the second number from the first.
  multiply(x, y): Multiplies two numbers.
  divide(x, y): Divides the first number by the second.
"""


def add(x, y):
 """
 Adds two numbers.
 Parameters
 ----------
 x : int or float
    The first number.
 y : int or float
    The second number.

 Returns
  int or float
    The sum of x and y.
 """
 return x + y
def subtract(x, y):
 """
 Subtracts the second number from the first.
Parameters
 ----------
 x : int or float
    The first number (minuend).
 y : int or float
    The second number (subtrahend).
 Returns
 -------
 int or float
    The difference between x and y.
 """
 return x – y
```

```
def multiply(x, y):
 """
 Multiplies two numbers.
Parameters
 ----------
 x : int or float
   The first number.
 y : int or float
   The second number.
Returns
 -------
 int or float
   The product of x and y.
 """
 return x * y
def divide(x, y):
 """
 Divides the first number by the second.

 Parameters
 x : int or float
   The first number (dividend).
 y : int or float
   The second number (divisor).

 Returns
 -------
 int or float
```

```
        The result of the division.

    Raises
    ------
    ZeroDivisionError
        If the divisor (y) is zero.
    """
    if y == 0:
        raise ZeroDivisionError("Cannot divide by zero")
    return x / y

# Example usage:
num1 = 10
num2 = 5

print(f"{num1} + {num2} = {add(num1, num2)}")
print(f"{num1} - {num2} = {subtract(num1, num2)}")
print(f"{num1} * {num2} = {multiply(num1, num2)}")
print(f"{num1} / {num2} = {divide(num1, num2)}")

# Example of division by zero
# try:
#     print(f"{num1} / 0 = {divide(num1, 0)}")
# except ZeroDivisionError as e:
#     print(e)
```

OUT PUT :



```
10 + 5 = 15
10 - 5 = 5
10 * 5 = 50
10 / 5 = 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.

TASK 4