AI ASSISTED CODING

**ASSIGNMENT – 9.3**

**NAME :** YUVARAJ REGULA

**HALL.NO :** 2403A52386

**BATCH.NO :** AIML-14

**PLATFORM USED :** GOOGLE COLLAB GEMINI AND perplexity.ai

**PROMPT 01 :**

genrate a python function to return sum of even and odd numbers in the given list.,and also generate a docstring describing the function.

**CODE , OUTPUT AND DOCSTRING : ( by GEMINI)**

def sum\_even\_odd(numbers):

  """R

  Calculates the sum of even and odd numbers in a given list.

  Args:

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

  Returns:

    A tuple containing two elements:

    - The sum of even numbers in the list.

    - The sum of odd numbers in the list.

  """

  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}")

**CODE , OUTPUT AND DOCSTRING : ( by PERPLEXITY.AI)**

def sum\_even\_odd(nums):

    """

    Return the sum of even numbers and the sum of odd numbers from a list.

    Parameters:

        nums (list[int | float]): A list of numeric values. Non-integers are treated by their integer parity (e.g., 2.0 is even). Non-numeric items should be filtered out by the caller.

    Returns:

        tuple[int | float, int | float]: A tuple (even\_sum, odd\_sum) where:

            - even\_sum is the sum of all elements with num % 2 == 0

            - odd\_sum is the sum of all elements with num % 2 == 1

    Notes:

        - Parity is determined using the modulo operator: num % 2 == 0 for even, num % 2 == 1 for odd. [1]

        - For performance and clarity, this implementation uses a single pass and boolean-to-int arithmetic. [1]

        - Python’s built-in sum() aggregates the totals efficiently once partials are accumulated. [2]

    Examples:

        >>> sum\_even\_odd([1, 2, 3, 4, 5, 6])

        (12, 9)

        >>> sum\_even\_odd([])

        (0, 0)

        >>> sum\_even\_odd([0, -1, -2, 7])

        (-2, 6)

    """

    even\_sum = 0

    odd\_sum = 0

    for num in nums:

        # Treat booleans as integers in arithmetic: True -> 1, False -> 0

        even\_sum += num \* (num % 2 == 0)

        odd\_sum += num \* (num % 2 == 1)

    return even\_sum, odd\_sum

**CODE EXPLANATION :**

* def sum\_even\_odd(numbers):: This line defines a function named sum\_even\_odd that takes one argument, numbers, which is expected to be a list.
* """Docstring""": This is a docstring, providing a brief explanation of what the function does, its arguments, and what it returns.
* even\_sum = 0 and odd\_sum = 0: These lines initialize two variables, even\_sum and odd\_sum, to zero. These variables will store the cumulative sum of even and odd numbers, respectively.
* for number in numbers:: This line starts a for loop that iterates through each number in the input numbers list.
* if number % 2 == 0:: Inside the loop, this line checks if the current number is even by using the modulo operator (%). If the remainder when divided by 2 is 0, the number is even.
* even\_sum += number: If the number is even, this line adds the number to the even\_sum.
* else:: If the if condition is false (the number is not even), the code inside the else block is executed.
* odd\_sum += number: If the number is odd, this line adds the number to the odd\_sum.
* return even\_sum, odd\_sum: After the loop finishes iterating through all the numbers, this line returns a tuple containing the final even\_sum and odd\_sum.

**PROMPT 02 :**python program for sru\_student class with attributes like name, roll no., hostel\_status and fee\_update method and display\_details method.AND ASLO add inline comments explaining each line/step

**CODE , OUTPUT AND DOCSTRING : ( by GEMINI)**

class sru\_student: # Define a class named sru\_student

    def \_\_init\_\_(self, name, roll\_no, hostel\_status): # Constructor to initialize the object with name, roll number, and hostel status

        self.name = name # Assign the provided name to the object's name attribute

        self.roll\_no = roll\_no # Assign the provided roll number to the object's roll\_no attribute

        self.hostel\_status = hostel\_status # Assign the provided hostel status to the object's hostel\_status attribute

        self.fees\_paid = 0 # Initialize fees\_paid attribute to 0

    def fee\_update(self, amount): # Define a method to update the fees paid

        self.fees\_paid += amount # Add the provided amount to the fees\_paid attribute

        print(f"Fees updated for {self.name}. Total fees paid: {self.fees\_paid}") # Print a confirmation message

    def display\_details(self): # Define a method to display the student's details

        print("Student Details:") # Print a header for the 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

# Example usage:

# student1 = sru\_student("Alice", "SRU123", "Resident") # Create an instance of the sru\_student class

# student1.display\_details() # Display the details of student1

# student1.fee\_update(5000) # Update the fees for student1

# student1.display\_details() # Display the updated details of student1

**CODE , OUTPUT AND DOCSTRING : ( by PERPLEXITY.AI)**

class SRUStudent:  # Define a class named SRUStudent to model a student entity [10][17]

    def \_\_init\_\_(self, name, roll\_no, hostel\_status, total\_fee):  # Constructor initializes instance attributes [10][17]

        self.name = name  # Store the student's name as an instance attribute [10][17]

        self.roll\_no = roll\_no  # Store the student's roll number [10][17]

        self.hostel\_status = bool(hostel\_status)  # Store hostel status as True/False for consistency [10][17]

        self.total\_fee = float(total\_fee)  # Store the current total fee as a float value [10][17]

    def fee\_update(self, amount, operation="add"):  # Define method to update fee; supports add/subtract operations [10][17]

        # amount: numeric value to add or subtract from total\_fee [10][17]

        # operation: "add" to increase fee, "subtract" to decrease fee [10][17]

        amount = float(amount)  # Ensure the amount is treated as a float for arithmetic [10][17]

        if operation == "add":  # If the chosen operation is add, increase total\_fee [10][17]

            self.total\_fee += amount  # Increment the fee by the specified amount [10][17]

        elif operation == "subtract":  # If the operation is subtract, decrease total\_fee [10][17]

            self.total\_fee -= amount  # Decrement the fee by the specified amount [10][17]

        else:  # For any other operation value, raise an error to signal misuse [10][17]

            raise ValueError("operation must be 'add' or 'subtract'")  # Inform the caller of valid operations [10][17]

    def display\_details(self):  # Define method to print student details in a readable format [10][17]

        status = "Hosteller" if self.hostel\_status else "Day Scholar"  # Convert boolean to human-readable label [10][17]

        print(f"Name       : {self.name}")  # Print the student's name [10][17]

        print(f"Roll No.   : {self.roll\_no}")  # Print the student's roll number [10][17]

        print(f"Residence  : {status}")  # Print hostel/day-scholar status [10][17]

        print(f"Total Fee  : ₹{self.total\_fee:.2f}")  # Print current total fee formatted to two decimals [10][17]

# Example usage (for quick testing):  # Inline example showing how to create and use the class [13]

if \_\_name\_\_ == "\_\_main\_\_":  # Standard entry-point guard for scripts [10][17]

    s1 = SRUStudent("Arun Kumar", "SRU23CS001", hostel\_status=True, total\_fee=85000)  # Create a student object with initial data [10][17]

    s1.display\_details()  # Show initial details [10][17]

    s1.fee\_update(5000, "add")  # Add a fee (e.g., lab charges) [10][17]

    s1.fee\_update(2000, "subtract")  # Subtract a waiver/scholarship amount [10][17]

    s1.display\_details()  # Show updated details after fee changes [10][17]

**CODE EXPLANATION :**

* **class sru\_student:**: This line defines a new class named sru\_student. Classes are blueprints for creating objects.
* **def \_\_init\_\_(self, name, roll\_no, hostel\_status):**: This is the constructor method (\_\_init\_\_). It's automatically called when you create a new sru\_student object. It takes name, roll\_no, and hostel\_status as arguments and initializes the object's attributes with these values. self refers to the instance of the class being created.
* **self.name = name**, **self.roll\_no = roll\_no**, **self.hostel\_status = hostel\_status**: These lines assign the values passed to the constructor to the object's attributes (name, roll\_no, and hostel\_status).
* **self.fees\_paid = 0**: This line initializes the fees\_paid attribute to 0 for every new student object.
* **def fee\_update(self, amount):**: This defines a method called fee\_update. It takes self (the object itself) and an amount as arguments.
* **self.fees\_paid += amount**: Inside fee\_update, this line adds the amount to the object's fees\_paid attribute, effectively updating the fees paid.
* **print(f"Fees updated for {self.name}. Total fees paid: {self.fees\_paid}")**: This line prints a confirmation message after updating the fees, showing the student's name and the new total fees paid.
* **def display\_details(self):**: This defines a method called display\_details. It takes self as an argument.
* **print(...)**: The lines within display\_details print the student's details (Name, Roll No., Hostel Status, and Fees Paid) in a formatted way.

**PROMPT 03 :**

GERATE A Python s functions , calculator: add, subtract, multiply, divide. AND ALSO generate a module-level docstring + individual function docstrings.

**CODE , OUTPUT AND DOCSTRING : ( by GEMINI)**

"""

A simple calculator module providing basic arithmetic operations.

This module contains functions for adding, subtracting, multiplying,

and dividing two numbers.

"""

def add(x, y):

  """

  Adds two numbers.

  Args:

    x: The first number.

    y: The second number.

  Returns:

    The sum of x and y.

  """

  return x + y

def subtract(x, y):

  """

  Subtracts the second number from the first.

  Args:

    x: The first number.

    y: The second number.

  Returns:

    The difference between x and y.

  """

  return x - y

def multiply(x, y):

  """

  Multiplies two numbers.

  Args:

    x: The first number.

    y: The second number.

  Returns:

    The product of x and y.

  """

  return x \* y

def divide(x, y):

  """

  Divides the first number by the second.

  Args:

    x: The numerator.

    y: The denominator.

  Returns:

    The result of the division.

  Raises:

    ValueError: If the denominator (y) is zero.

  """

  if y == 0:

    raise ValueError("Cannot divide by zero!")

  return x / y

# Example usage:

# print(f"2 + 3 = {add(2, 3)}")

# print(f"5 - 2 = {subtract(5, 2)}")

# print(f"4 \* 6 = {multiply(4, 6)}")

# print(f"10 / 2 = {divide(10, 2)}")

# try:

#   print(f"5 / 0 = {divide(5, 0)}")

# except ValueError as e:

#   print(e)

**CODE , OUTPUT AND DOCSTRING : ( by PERPLEXITY.AI)**

"""

calculator.py

A simple calculator module providing basic arithmetic operations:

addition, subtraction, multiplication, and division.

Functions:

- add(a, b): Return the sum of a and b.

- subtract(a, b): Return the difference a - b.

- multiply(a, b): Return the product a \* b.

- divide(a, b): Return the quotient a / b with zero-division protection.

Usage:

    from calculator import add, subtract, multiply, divide

    result = add(10, 5)

    print(result)  # 15

Notes:

- All operations expect numeric inputs (int or float).

- divide() raises ZeroDivisionError if b == 0 (explicit check).

- Designed for teaching and reuse in small projects or assignments.

"""

def add(a: float, b: float) -> float:

    """

    Return the sum of two numbers.

    Parameters:

        a (float): First addend (int or float accepted).

        b (float): Second addend (int or float accepted).

    Returns:

        float: The arithmetic sum a + b.

    Examples:

        >>> add(3, 7)

        10

        >>> add(2.5, 0.5)

        3.0

    """

    return a + b

def subtract(a: float, b: float) -> float:

    """

    Return the difference of two numbers (a minus b).

    Parameters:

        a (float): Minuend (int or float accepted).

        b (float): Subtrahend (int or float accepted).

    Returns:

        float: The result a - b.

    Examples:

        >>> subtract(10, 4)

        6

        >>> subtract(3.0, 7.5)

        -4.5

    """

    return a - b

def multiply(a: float, b: float) -> float:

    """

    Return the product of two numbers.

    Parameters:

        a (float): First factor (int or float accepted).

        b (float): Second factor (int or float accepted).

    Returns:

        float: The product a \* b.

    Examples:

        >>> multiply(6, 7)

        42

        >>> multiply(1.5, 4)

        6.0

    """

    return a \* b

def divide(a: float, b: float) -> float:

    """

    Return the quotient of two numbers (a divided by b).

    Parameters:

        a (float): Dividend (int or float accepted).

        b (float): Divisor (int or float accepted). Must not be zero.

    Returns:

        float: The result a / b.

    Raises:

        ZeroDivisionError: If b == 0.

    Examples:

        >>> divide(8, 2)

        4.0

        >>> divide(7.5, 2.5)

        3.0

        >>> divide(5, 0)

        Traceback (most recent call last):

        ...

        ZeroDivisionError: division by zero is not allowed

    """

    if b == 0:

        raise ZeroDivisionError("division by zero is not allowed")

    return a / b

**CODE EXPLANATION :**

* """A simple calculator module...""": This is the module-level docstring. It explains the overall purpose of the code in this module.
* def add(x, y):: This defines a function named add that takes two arguments, x and y.
* """Adds two numbers.""": This is the docstring for the add function, explaining what it does, its arguments (Args), and what it returns (Returns).
* return x + y: This line calculates the sum of x and y and returns the result.
* def subtract(x, y):: This defines the subtract function, similar to add, taking x and y.
* """Subtracts the second number...""": This is the docstring for the subtract function.
* return x - y: This line calculates the difference between x and y and returns it.
* def multiply(x, y):: This defines the multiply function, taking x and y.
* """Multiplies two numbers.""": This is the docstring for the multiply function.
* return x \* y: This line calculates the product of x and y and returns it.
* def divide(x, y):: This defines the divide function, taking x and y.
* """Divides the first number...""": This is the docstring for the divide function, also mentioning the Raises exception for division by zero.
* if y == 0:: This line checks if the denominator y is equal to 0.
* raise ValueError("Cannot divide by zero!"): If y is 0, this line raises a ValueError with a specific message, indicating that division by zero is not allowed.
* return x / y: If y is not 0, this line performs the division and returns the result.