Lab Assignment -9.3

**Name:** Bhavaneesh.A

**Roll No:**  2403A52350

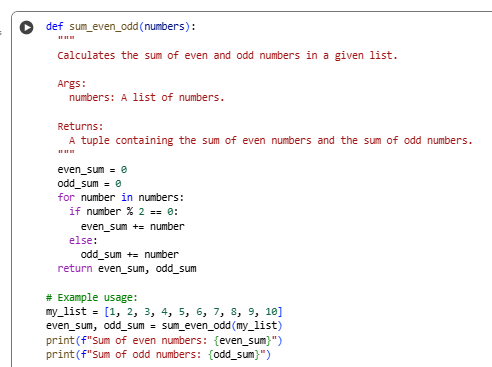
**Batch:** Ai 13

**Task – 1: Using Gemini**

**Prompt:**

Write python function to return sum of even and odd numbers in the given list. using docstring.

**Code:**



**Output:**



**Explanation:**

This code defines a Python function called sum\_even\_odd that takes a list of numbers as input and calculates the sum of the even and odd numbers within that list.

* def sum\_even\_odd(numbers): : This line defines the function named sum\_even\_odd and indicates that it accepts one argument, which will be referred to as numbers within the function.
* """ ... """ : This is a docstring, which provides a brief description of what the function does, its arguments (Args:), and what it returns (Returns:). This is good practice for documenting your code.
* even\_sum = 0 and odd\_sum = 0 : These lines initialize two variables, even\_sum and odd\_sum, to zero. These variables will be used to store the running total of even and odd numbers respectively.
* for number in numbers: : This is a for loop that iterates through each element in the input list numbers. In each iteration, the current element is assigned to the variable number.
* 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.
* even\_sum += number : If the number is even, this line adds the current number to the even\_sum.
* else: : This is the alternative case if the number is not even (meaning it's odd).
* odd\_sum += number : If the number is odd, this line adds the current number to the odd\_sum.
* return even\_sum, odd\_sum : After the loop has finished iterating through all the numbers in the list, this line returns the final values of even\_sum and odd\_sum as a tuple.

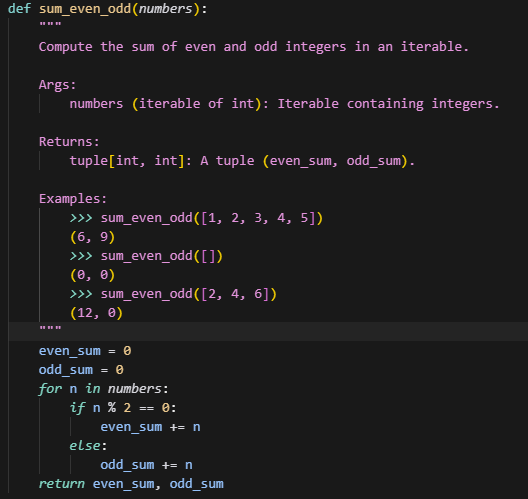
The code then demonstrates how to use the function with an example list my\_list and prints the calculated sums of even and odd numbers.

**Task – 1: Using Cursor**

**Prompt:**

Write python function to return sum of even and odd numbers in the given list. using docstring.

**Code:**



**Differences:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Gemini** | **Cursor** |
| **Docstring Style** | **Google-style** | **NumPy / reStructuredText-style** |
| **Type Hints in Docstring** | **❌ Not included** | **✅ Included (Iterable[int], tuple[int, int])** |
| **Examples** | **Separate usage section below the function** | **Embedded in docstring using doctest (>>>)** |
| **Target Audience** | **Beginners, learners** | **Professionals, contributors, automated testing** |
| **Readability** | **More human-friendly** | **More formal & structured** |

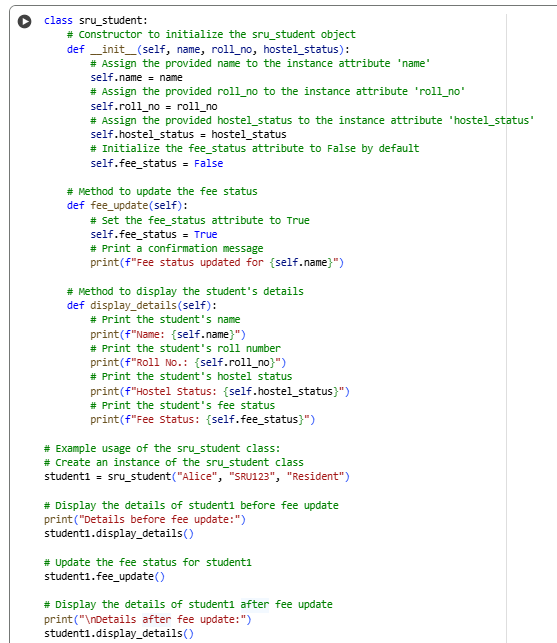
**Task – 2: Using Gemini**

**Prompt :**

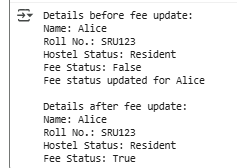
I have written a Python program for an sru\_student class with attributes like name, roll\_no, hostel\_status, a method fee\_update(), and a method display\_details().

1. Please add **inline comments** explaining each line/step of the code.
2. Keep the comments clear, beginner-friendly, and precise.
3. Do not rewrite the code structure, only add comments.

**Code:**



**Output:**



**Explanation:**

This code defines a Python class named sru\_student. A class is a blueprint for creating objects (instances) that have specific attributes (data) and methods (functions).

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 a special method that gets 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 the parameters that you pass when creating a new sru\_student object.
  + self.name = name, self.roll\_no = roll\_no, self.hostel\_status = hostel\_status: These lines assign the values passed during object creation to the instance attributes of the object.
  + self.fee\_status = False: This line initializes the fee\_status attribute to False by default for every new student object.
* def fee\_update(self):: This defines a method named fee\_update.
  + self: Refers to the instance of the class on which the method is called.
  + self.fee\_status = True: This line changes the fee\_status attribute of the specific sru\_student object to True.
  + print(f"Fee status updated for {self.name}"): This line prints a confirmation message indicating which student's fee status has been updated.
* def display\_details(self):: This defines a method named display\_details.
  + self: Refers to the instance of the class on which the method is called.
  + The print statements within this method display the values of the name, roll\_no, hostel\_status, and fee\_status attributes for the specific sru\_student object.

The code then shows an example of how to use the sru\_student class:

* student1 = sru\_student("Alice", "SRU123", "Resident"): This line creates a new instance of the sru\_student class named student1 and initializes it with the provided name, roll number, and hostel status.
* The subsequent print statements and calls to student1.display\_details() and student1.fee\_update() demonstrate how to access the attributes and call the methods of the student1 object.

**Task – 2: Using Cursor**



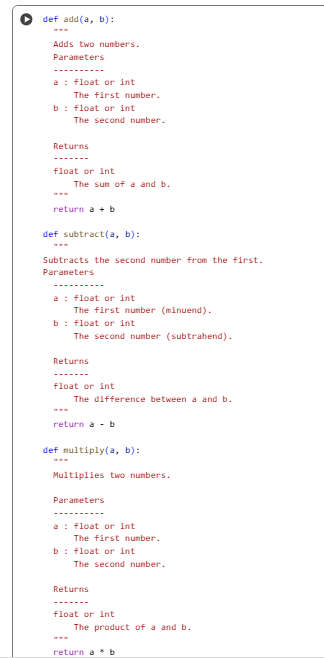
**Differences:**

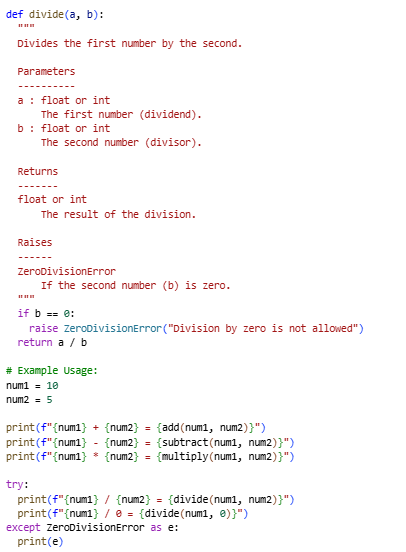
|  |  |  |
| --- | --- | --- |
| **Aspect** | **Gemini** | **Cursor** |
| **Documentation** | Inline comments before/inside methods | File-level docstring + method docstrings (professional style) |
| **Attributes** | name, roll\_no, hostel\_status, fee\_status (boolean flag only) | name, roll\_no, hostel\_status, total\_fee (numeric, cumulative) |
| **Fee Handling** | Just sets fee\_status = True (no actual calculation) | Computes total fee from base\_fee, additional\_fee, and hostel\_fee |
| **Methods** | fee\_update() → toggles flag display\_details() → prints details | fee\_update() → dynamically updates and returns fee display\_details() → formatted, professional output |
| **Example Usage** | Creates one student (Alice) Shows details before & after fee update | Uses if \_\_name\_\_ == "\_\_main\_\_": Creates multiple students (day scholar & hostel) Demonstrates multiple fee updates |
| **Code Structure** | Simple, beginner-friendly | Clean, modular, production-ready |
| **Target Audience** | Beginners learning classes | Professional/academic use, real-world scenarios |

**Task – 3: Using Gemini**

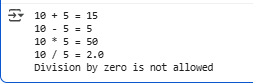
**Prompt:**

**Code:**





**Output :**



**Explanation:**

This code defines four basic mathematical functions: add, subtract, multiply, and divide. Each function takes two numbers as input and performs the corresponding operation. The docstrings for each function follow the NumPy style, which is a common convention for documenting Python code, especially in scientific computing.

Here's a breakdown of each function:

* add(a, b):
  + This function takes two parameters, a and b.
  + The docstring explains that it adds two numbers and specifies the types and descriptions of the parameters (Parameters) and the return value (Returns).
  + It simply returns the sum of a and b.
* subtract(a, b):
  + This function takes two parameters, a and b.
  + The docstring explains that it subtracts b from a, again specifying the parameters and return value.
  + It returns the difference between a and b.
* multiply(a, b):
  + This function takes two parameters, a and b.
  + The docstring explains that it multiplies two numbers and specifies the parameters and return value.
  + It returns the product of a and b.
* divide(a, b):
  + This function takes two parameters, a and b.
  + The docstring explains that it divides a by b, specifies the parameters and return value, and importantly, notes that it Raises a ZeroDivisionError if b is zero.
  + Inside the function, it checks if b is equal to 0.
  + If b is 0, it raises a ZeroDivisionError with a descriptive message.
  + If b is not 0, it returns the result of dividing a by b.

The code then demonstrates how to use these functions:

* num1 = 10 and num2 = 5: These lines initialize two variables with example numbers.
* The print statements show the results of calling the add, subtract, and multiply functions with num1 and num2.
* The try...except ZeroDivisionError block is used to handle the potential error when dividing by zero.
  + try:: This block contains the code that might raise an exception.
  + print(f"{num1} / {num2} = {divide(num1, num2)}"): This line calls the divide function with num1 and num2 and prints the result.
  + print(f"{num1} / 0 = {divide(num1, 0)}"): This line attempts to call the divide function with 0 as the second argument, which will raise a ZeroDivisionError.
  + except ZeroDivisionError as e:: This block catches the ZeroDivisionError if it occurs in the try block. The error object is assigned to the variable e.
  + print(e): This line prints the error message associated with the ZeroDivisionError.

**Task – 3: Using Cursor**

**Code:**



**Differences:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Gemini** | **Cursor** |
| **Docstring style** | Short docstrings with Parameters and Returns only. | NumPy-style detailed docstrings with Parameters, Returns, Examples, and Raises. |
| **Examples** | No usage examples inside docstrings. | Each function has examples showing expected input/output. |
| **Return type** | Specified as int in docstrings. | Specified as float in docstrings. |
| **Division function** | Uses if b == 0: return "Error: Division by zero is not allowed" (returns string). | Uses if b == 0: raise ZeroDivisionError("Cannot divide by zero.") (raises exception). |
| **Function outputs** | All functions return results directly (int). Division may return a string (error). | All functions return numeric results (float). Division raises error instead of returning string. |
| **Main section** | Prints results with labels like add(10, 5) = ... and subtract(10, 5) = .... | Similar prints, but also includes a comment # Simple demo of calculator functions. |
| **Code formatting** | Plain formatting, minimal spacing. | More structured formatting with blank lines, indentation, and color syntax (possibly from a modern editor like VS Code / Jupyter). |
| **Error handling** | Error is handled gracefully with a string message. | Error is handled via Python’s built-in exception (ZeroDivisionError). |
| **Consistency** | Return types inconsistent (sometimes int, sometimes error string). | Consistent return types (float or raises error). |
| **Readability** | Simpler, beginner-friendly. | Professional, more Pythonic and closer to production-quality code. |