**Use Case for the Employee Class Code:**

The provided code defines an Employee class that represents employees with attributes such as name, age, and salary. It also provides methods to retrieve employee information, increase their salary, and determine if they are elderly. The code also demonstrates the use of map() and filter() functions to perform operations on a list of employees.

Use Case: Managing Employee Information and Salary

Create and Store Employee Information:

Instantiate Employee objects with relevant information such as name, age, and salary.

Add the created employee objects to a list, employees, to store their information.

Increase Salary of Employees:

Utilize the map() function to iterate over the employees list.

Apply the get\_salary\_increase() method on each employee object, passing a percentage increase (e.g., 10%) as an argument.

The salary of each employee will be increased by the provided percentage.

Retrieve Elderly Employees:

Use the filter() function to iterate over the employees list.

Apply the is\_elderly() method on each employee object to check if they are elderly.

Create a new list, elderly\_employees, containing only the employee objects that are considered elderly (age >= 60).

The provided code allows for managing and manipulating employee data. By using the Employee class and its methods, you can retrieve employee information, modify salaries, and identify elderly employees efficiently. These functionalities can be utilized in various HR systems, payroll management systems, or employee analytics applications.

**Problem Statement**

The existing code defines an Employee class that represents employees with their name, age, and salary information. It also provides methods to retrieve employee details, increase their salary, and determine if they are considered elderly. However, there are several issues and limitations in the current implementation that need to be addressed.

1. Inconsistent Constructor Method:

The constructor method of the Employee class is misspelled as \_init\_ instead of \_\_init\_\_. This leads to the method not being called properly when creating new instances of the class.

1. Lack of Encapsulation and Data Validation:

The class attributes name, age, and salary are directly accessible from outside the class, which violates the principles of encapsulation. There are no checks or validations in place to ensure the correctness of the provided input values.

1. Inefficient Salary Increase Calculation:

The get\_salary\_increase() method calculates the salary increase using basic arithmetic operations. However, it doesn't account for rounding or formatting the resulting salary to a proper decimal place.

1. Inaccurate Elderly Determination:

The is\_elderly() method determines if an employee is considered elderly based on a fixed threshold of age 60. This threshold may not be universally applicable and might need to be customizable or configurable.

1. Limited Error Handling:

The current code lacks proper error handling mechanisms. If invalid input values are provided or operations fail, there are no specific error messages or exception handling procedures to handle such scenarios gracefully.

1. Lack of Employee Management Functionality:

The code only focuses on basic operations such as retrieving employee information, increasing salaries, and determining elderly employees. However, it lacks more advanced functionalities commonly associated with employee management systems, such as adding new employees, updating employee details, or removing employees from the list.

**Code for Employee**

class Employee:

def \_\_init\_\_(self, name, age, salary):

self.name = name

self.age = age

self.salary = salary

def get\_name(self):

return self.name

def get\_age(self):

return self.age

def get\_salary(self):

return self.salary

def get\_salary\_increase(self, percentage):

self.salary += (self.salary \* percentage) / 100

def is\_elderly(self):

if self.age >= 60:

return True

else:

return False

employees = [Employee("John", 45, 50000), Employee("Mary", 65, 60000), Employee("Bob", 55, 70000)]

# Increase the salary of each employee by 10%

list(map(lambda x: x.get\_salary\_increase(10), employees))

# Obtain a list of only the elderly employees

elderly\_employees = list(filter(lambda x: x.is\_elderly(), employees))