**Use case: Salary Increase and Elderly Employee Identification**

In a company, there is a list of employees with their names, ages, and salaries. The HR department wants to perform two operations on the employee data: increase the salary of each employee by 10% and identify the elderly employees (those with an age of 60 or greater).

The code snippet above represents a solution to this use case. Here's how it works:

1. The employees list is created, containing four Employee objects with different attributes (name, age, and salary).
2. Using the map function, the salary of each employee is increased by 10% by applying a lambda function that calls the get\_salary\_increase method on each employee.
3. The filter function is used to obtain a list of only the elderly employees by applying a lambda function that checks if the employee's age is 60 or greater.
4. The filtered list of elderly employees is stored in the elderly\_employees variable.
5. Finally, the details (name, age, and salary) of each elderly employee are printed using a loop.

This use case demonstrates how the Employee class and the map and filter functions can be utilized to efficiently manipulate and analyze employee data in a company setting.

**Problem Statement: Employee Salary Increase and Elderly Employees**

You are tasked with developing a Python program to handle employee data for a company. The program should provide functionality to increase the salary of each employee by a given percentage and identify the elderly employees based on their age.

You need to implement an Employee class with the following attributes and methods: Attributes:

name (str): The name of the employee.

age (int): The age of the employee.

salary (float): The salary of the employee.

Methods:

\_\_init\_\_(self, name, age, salary): Initializes a new employee with the given name, age, and salary.

get\_name(self): Returns the name of the employee.

get\_age(self): Returns the age of the employee.

get\_salary(self): Returns the salary of the employee.

get\_salary\_increase(self, percentage): Increases the salary of the employee by the specified percentage.

is\_elderly(self): Returns True if the employee's age is 60 or greater, and False otherwise.

You also need to provide a sample code snippet that demonstrates the usage of the class by creating a list of Employee objects and performing various operations on them using map and filter. Specifically, you should use map to increase the salary of each employee by 10% and use filter to obtain a list of only the elderly employees (those with an age of 60 or greater).

Ensure that the implementation is modular, efficient, and follows best practices for object-oriented programming in Python.

Your program should be able to handle any number of employees and provide accurate results for salary increases and identification of elderly employees.

Write a detailed problem statement that clearly explains the requirements and expected behavior of the program.

**Code for Employee Salary Increase and Elderly Employees :**

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 \*= (1 + percentage / 100)

def is\_elderly(self):

return self.age >= 60

employees = [

Employee("John Doe", 45, 5000.0),

Employee("Jane Smith", 55, 6000.0),

Employee("Bob Johnson", 65, 7000.0),

Employee("Alice Williams", 70, 8000.0),

]

salary\_increase\_percentage = 10

list(map(lambda emp: emp.get\_salary\_increase(salary\_increase\_percentage), employees))

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

for emp in elderly\_employees:

print("Name:", emp.get\_name())

print("Age:", emp.get\_age())

print("Salary:", emp.get\_salary())

print()

**Explanation of code :**

The Employee class represents an employee of a company and has the following attributes:

* name (str): The name of the employee.
* age (int): The age of the employee.
* salary (float): The salary of the employee.

The class also provides the following methods:

* \_\_init\_\_(self, name, age, salary): This is the constructor method that initializes a new employee with the given name, age, and salary.
* get\_name(self): Returns the name of the employee.
* get\_age(self): Returns the age of the employee.
* get\_salary(self): Returns the salary of the employee.
* get\_salary\_increase(self, percentage): Increases the salary of the employee by the specified percentage. It takes a percentage as a parameter, converts it to a decimal value, and updates the salary attribute accordingly.
* is\_elderly(self): Returns True if the employee's age is 60 or greater, and False otherwise.

1. The sample code snippet demonstrates the usage of the Employee class by performing operations on a list of Employee objects.
2. A list called employees is created, containing four Employee objects with different attributes (name, age, and salary).
3. The map function is used to increase the salary of each employee in the employees list by 10%. A lambda function is applied to each employee, calling the get\_salary\_increase method with the specified percentage.
4. The filter function is used to create a new list called elderly\_employees, which contains only the elderly employees from the original list. A lambda function is applied to each employee, checking if their age is 60 or greater using the is\_elderly method.
5. Finally, a loop is used to iterate over the elderly\_employees list and print the details (name, age, and salary) of each elderly employee.

The code snippet demonstrates how the Employee class can be used to manage employee data. It showcases the use of map to apply a salary increase to each employee and filter to obtain a subset of employees based on a condition. The output displays the details of the elderly employees after the salary increase.

**Output for code :**

Name: Alice Williams

Age: 70

Salary: 8800.0