NEURAL NETWORKS & DEEP LEARNING - ICP3

NAME: ABRAR PASHA MOHAMMED

700#: 700744984

GITHUB LINK: https://github.com/Abrar2456/ICP3.git

1. Create a class Employee and then do the following

• Create a data member to count the number of Employees

• Create a constructor to initialize name, family, salary, department

• Create a function to average salary

• Create a Fulltime Employee class and it should inherit the properties of Employee class

• Create the instances of Fulltime Employee class and Employee class and call their member functions.

- ➔ Initially, we have created the Employee class & Full-Time Employee class which has different parameters.
- ➔ The Employee class has attributes such as name, family, salary, and department, along with methods to calculate the average salary and display employee details.
- ➔ The FulltimeEmployee class is a subclass of Employee, inheriting its properties.
- ➔ The code then creates instances of both classes, emp1 and emp2 of type Employee, and of type FulltimeEmployee.
- ➔ It initializes their attributes and calls methods to display employee details and the total number of employees.
- ➔ The average salary of all employees is also calculated and printed.
- ➔ The code demonstrates the use of classes, inheritance, object instantiation, and class methods to manage and display employee information.

In [8]:
```
class Employee:#Class for Employee objects
    employeeCount=0 #To count number of employee objects
    salarySum=0 #sum of salaries to find average of the sum
    def __init__(self,name,family,salary,department): #constructor for Employee as per the requirements
        self.name = name
        self.family = family
        self.salary = salary
        self.department = department
        Employee.employeeCount += 1
        Employee.salarySum+=self.salary
    def avg(self):
        return Employee.salarySum/Employee.employeeCount
    def displayEmployee(self):
        print("Name :", self.name, ", Family:", self.family , ", Salary:", self.salary, ", Department:",self.department)

emp1 = Employee("Abrar Pasha","Mohammed",40000,"Computer Science")#Employee objects creation with intialization
emp2 = Employee("Azeem Pasha","Mohammed",50000,"Electronics")
print("avg:",emp2.avg())#calling Average function

class FulltimeEmployee(Employee):
    def __init__(self, name, family, salary, department):
        super().__init__(name, family, salary, department)

fte=FulltimeEmployee("Shahista","Farheen",90000,"Medic")#FultimeEmployee object
fte.displayEmployee()
```

```
avg: 45000.0
Name : Shahista , Family: Farheen , Salary: 90000 , Department: Medic
```

PTO

2. Using NumPy create random vector of size 20 having only float in the range 1-20. Then reshape the array to 4 by 5 Then replace the max in each row by 0 (axis=1) (you can NOT implement it via for loop)

➔ A NumPy vector is generated using the arange function, creating a float array with values ranging from 1 to 20 (inclusive).
➔ The shape of the vector is altered to form a 4x5 matrix using the reshape function, resulting in a structure with 4 rows and 5 columns.
➔ The vec.max(axis=1) operation identifies the maximum value along each row (axis=1) in the matrix.
➔ np.isin(vec, vec.max(axis=1)) creates a boolean matrix by checking if each element in the matrix is equal to the maximum value in its corresponding row.
➔ np.where("condition", 0, vec) is used to replace elements where the specified condition is true (in this case, where an element equals the maximum value in its row) with 0, while leaving other elements unaltered. This effectively replaces the maximum values in each row with zero.

```
In [9]: import numpy as np #importing the numpy library
        vector=np.arange(1,26,dtype=float)
        print(vector)
        vector=vector.reshape(5,5)
        print(vector)
        vector=np.where(np.isin(vector,vector.max(axis=1)),0,vector)
        vector

        [ 1.  2.  3.  4.  5.  6.  7.  8.  9. 10. 11. 12. 13. 14. 15. 16. 17. 18.
         19. 20. 21. 22. 23. 24. 25.]
        [[ 1.  2.  3.  4.  5.]
         [ 6.  7.  8.  9. 10.]
         [11. 12. 13. 14. 15.]
         [16. 17. 18. 19. 20.]
         [21. 22. 23. 24. 25.]]

Out[9]: array([[ 1.,  2.,  3.,  4.,  0.],
               [ 6.,  7.,  8.,  9.,  0.],
               [11., 12., 13., 14.,  0.],
               [16., 17., 18., 19.,  0.],
               [21., 22., 23., 24.,  0.]])
```