Github Link : https://github.com/AnishKoppula1/NeuralAssignment3

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.

```python
#Employee Class
class Employee:
    num_of_employees = 0  #data member that counts the number of Employees

    # constructor that initializes name, family, salary, department
    def __init__(self, name, family, salary, department):
        self.name = name
        self.family = family
        self.salary = salary
        self.department = department
        Employee.num_of_employees += 1   #Incrementing the employee count on instance creation

    #a function to average salary that accepts Employee array and returns average
    @staticmethod
    def employees_average_salary(employees):
        employee_salaries = [emp.salary for emp in employees] #creating an array of Salaries using list comprehension
        sum_of_salaries = sum(employee_salaries)
        employee_count = len(employees)
        return sum_of_salaries / employee_count


#FulltimeEmployee Class
class FulltimeEmployee(Employee):
    # constructor that initializes name, family, salary, department
    def __init__(self, name, family, salary, department):
        super().__init__(name, family, salary, department) #calling EMployee class Constructor

# Creating instances of Employee and FulltimeEmployee classes
employee_1 = Employee("Anish", "Koppula", 30000, "Associate Software Developer")
employee_2 = Employee("Rohit", "Sharma", 100000, "Manager")
employee_3 = FulltimeEmployee("Virat", "Kohli", 50000, "Senior Developer")
employee_4 = FulltimeEmployee("Arunab", "Koppula", 7000000, "CEO")

#creating a list using the Employee and FulltimeEmployee class instances
employees = [employee_1, employee_2, employee_3, employee_4]

#calling datamember using Employee Class
print("No of Employess : " + str(FulltimeEmployee.num_of_employees))

# Calling member functions using FulltimeEmployee class
print("Average salary of all employees : " + str(Employee.employees_average_salary(employees)))

```

Output:

```
In [9]: runfile('C:/Users/koppu/ICP3_1py.py', wdir='C:/Users/
koppu')
No of Employess : 4
Average salary of all employees : 1795000.0
```

2.Numpy

- 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)

```python
import numpy as np

starting_range = 1    #variable indicating starting range of vector elements
ending_range = 20     #variable indicating ending range of vector elements
no_of_elements = 20   #variable indicating no of elements the vector should hold

# Creating a random vector of size 20 with floats in the range 1-20
random_vector = np.random.uniform(starting_range, ending_range, no_of_elements)

#printing the random vector
print("random vector : \n" + str(random_vector))

# Reshape the array to 4 by 5
new_vector = random_vector.reshape(4, 5)

#printing the reshaped vector
print("reshaped vector : \n" + str(new_vector))

# getting indexes in each row where the element is maximum (axis=1)
max_index = np.argmax(new_vector, axis=1)

#creating new vector with no of rows like [0 1 2 3] so that we can use pairing for replacing elements
rows_vector = np.arange(new_vector.shape[0])

#replacing vector element of each row's maxindex with zero by creating pair of indexes
new_vector[rows_vector, max_index] = 0

#printing the reshaped vector after replacing max with 0
print("replaced vector : \n" + str(new_vector))
```

Output:

```
In [8]: runfile('C:/Users/koppu/ICP3_2py.py', wdir='C:/Users/koppu')
random vector :
[19.84132744  7.11976711  8.31997507  7.34259012 19.26353272  4.10775606
 19.10378524  7.21370329  6.30042099  6.90178822 17.0908398   7.58928681
  5.04620596 15.90556858  8.88882396  9.04684704  3.68604446 10.47760004
  4.63713805 18.63790272]


reshaped vector :
[[19.84132744  7.11976711  8.31997507  7.34259012 19.26353272]
 [ 4.10775606 19.10378524  7.21370329  6.30042099  6.90178822]
 [17.0908398   7.58928681  5.04620596 15.90556858  8.88882396]
 [ 9.04684704  3.68604446 10.47760004  4.63713805 18.63790272]]


replaced vector :
[[ 0.          7.11976711  8.31997507  7.34259012 19.26353272]
 [ 4.10775606  0.          7.21370329  6.30042099  6.90178822]
 [ 0.          7.58928681  5.04620596 15.90556858  8.88882396]
 [ 9.04684704  3.68604446 10.47760004  4.63713805  0.        ]]
```