

ICP-2

S.Ramyasri (7007727100)

GitHub Link: <https://github.com/Ramyasri0123/ICP2>

Video Link: <https://drive.google.com/file/d/1s18o-9QlbtuEIUK9K1yiupaYTXZcVqlw/view?usp=sharing>

Q-1 Code

```
ICP2.ipynb X
C:\Users> Ramya > Downloads > ICP2.ipynb > class Employee:
+ Code + Markdown ...
class Employee:
    employee_count = 0
    total_salary = 0

    def __init__(self, name, family, salary, department):
        self.name = name
        self.family = family
        self.salary = salary
        self.department = department
        Employee.employee_count += 1
        Employee.total_salary += salary

    @classmethod
    def average_salary(cls):
        if cls.employee_count > 0:
            return cls.total_salary / cls.employee_count
        return 0

    def display_employee(self):
        return f"Employee: {self.name}, Department: {self.department}, Salary: {self.salary}"

class FulltimeEmployee(Employee):
    ft_employee_count = 0
    ft_total_salary = 0

    def __init__(self, name, family, salary, department):
        super().__init__(name, family, salary, department)
        FulltimeEmployee.ft_employee_count += 1
        FulltimeEmployee.ft_total_salary += salary

    @classmethod
    def average_salary(cls):
        if cls.ft_employee_count > 0:
            return cls.ft_total_salary / cls.ft_employee_count
        return 0

    def display_employee(self):
        return f"Fulltime Employee: {self.name}, Department: {self.department}, Salary: {self.salary}"

# Creating instances
emp1 = Employee("harsha", "karthik", 60000, "HR")
emp2 = Employee("harsha", "karthik", 70000, "Finance")
ft_emp1 = FulltimeEmployee("Alice", "Smith", 80000, "IT")
ft_emp2 = FulltimeEmployee("karthik", "harsha", 90000, "Marketing")

# Displaying information
print(emp1.display_employee())
print(emp2.display_employee())
print(ft_emp1.display_employee())
print(ft_emp2.display_employee())

# Displaying average salaries and counts
print(f"Average Salary of Employees: {Employee.average_salary()}")
print(f"Number of Employees: {Employee.employee_count}")
print(f"Average Salary of Fulltime Employees: {FulltimeEmployee.average_salary()}")
print(f"Number of Fulltime Employees: {FulltimeEmployee.ft_employee_count}")
```

Explanation:

Employee class:

- Any employee is represented by the Employee Class.
- The sum of all employees' pay is counted.
- Every worker has a department, name, family, and pay.
- The average wage of all employees can be computed and displayed.
- Shows personal information about each employee.

Fulltime Employee Class (Inherits from Employee):

- The total number of full-time employees and their combined pay are counted independently.
- Features from the Employee class are carried over, and full-time specific tracking is added.
- The average wage of full-time LME employees can be computed and displayed. Individual full-time employee details are displayed.

Creating and Using Instances: -

- Specific details are created for both general and full-time employees.
- Each employee's details are displayed by the application.
- The average incomes for each category are calculated and displayed, along with their respective counts.

Output:

```
... Employee: harsha, Department: HR, Salary: 60000
Employee: harsha, Department: Finance, Salary: 70000
Fulltime Employee: Alice, Department: IT, Salary: 80000
Fulltime Employee: karthik, Department: Marketing, Salary: 90000
Average Salary of Employees: 75000.0
Number of Employees: 4
Average Salary of Fulltime Employees: 85000.0
Number of Fulltime Employees: 2
```

Q2

Code:

1. Creating a Random Vector:

```
[ ] import numpy as np

# Create random vector of size 20 with floats in the range 1-20
random_vector = np.random.uniform(1, 20, 20)

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

# Replace the max in each row by 0
reshaped_array[reshaped_array == reshaped_array.max(axis=1, keepdims=True)] = 0

reshaped_array
```

- `np.random.uniform(1, 20, 20)`: This line generates a random vector of size 20. The values are floating-point numbers, uniformly distributed between 1 and 20. The `uniform` function is used to ensure each number has an equal probability of being selected within this range.

2. Reshaping the Vector to a 4x5 Array:

- `random_vector.reshape(4, 5)`: The `reshaped_array` is the original vector reshaped into a 2D array with 4 rows and 5 columns. Reshaping doesn't change the data, it just changes how it's organized. The first parameter (4) indicates the number of rows, and the second (5) the number of columns.

3. Replacing the Maximum Value in Each Row with 0:

- `reshaped_array.max(axis=1, keepdims=True)`: This part finds the maximum value in each row of the array. `axis=1` specifies that the operation should be done row-wise (as opposed to column-wise with `axis=0`). `keepdims=True` ensures that the output has the same number of dimensions as the input, which is necessary for the next step.

- `reshaped_array == ...`: This creates a boolean array where each element is `True` if it's equal to the row's maximum value and `False` otherwise.
- `reshaped_array[...] = 0`: This replaces every element in `reshaped_array` that corresponds to `True` in the boolean array with 0. Essentially, it sets the maximum value in each row to 0.

Output:

```
array([[14.70134197, 13.03554151, 7.25405228, 0.        , 6.23386914],
       [ 5.45423432,  5.7069759 , 0.        ,  2.63847477,  2.03058003],
       [14.92580646,  6.8243149 ,  2.95644387, 12.89565488,  0.        ],
       [ 5.92459878,  6.89802205,  4.47456801,  7.45444271,  0.        ]])
```