Python Classes

Regular, class and static methods

Regular methods:

```
class Employee:
    raise_amt = 1.04
    no_of_employees = 0

    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary
        self.email = self.fname + '.' + self.lname + '@gmail.com'
        Employee.no_of_employees += 1

    def fullname(self):
        return f"{self.fname} {self.lname}"

    def pay_raise(self):
        self.salary = int(self.salary * self.raise_amt)

emp1 = Employee("Bala", "Muthu", 30000)
emp2 = Employee("Ramesh", "Raghul", 50000)

print(Employee.no_of_employees)
```

Note: the methods “fullname” and “pay_raise” are regular methods. They always takes instance as their first argument (“self”)

Class methods:

```
class Employee:
    raise_amt = 1.04
    no_of_employees = 0

    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary
        self.email = self.fname + '.' + self.lname + '@gmail.com'
        Employee.no_of_employees += 1

    def fullname(self):
        return f"{self.fname} {self.lname}"

    def pay_raise(self):
        self.salary = int(self.salary * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        pass

emp1 = Employee("Bala", "Muthu", 30000)
emp2 = Employee("Ramesh", "Raghul", 50000)

print(Employee.raise_amt)
print(emp1.raise_amt)
print(emp2.raise_amt)
```

The class methods should have a decorator “@classmethod” before the method definition.

When you notice, the class method takes “cls” as the first argument.

Output:

1.04
1.04
1.04

```python
class Employee:
    raise_amt = 1.04
    no_of_employees = 0

    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary
        self.email = self.fname + '.' + self.lname + '@gmail.com'
        Employee.no_of_employees += 1

    def fullname(self):
        return f"{self.fname} {self.lname}"

    def pay_raise(self):
        self.salary = int(self.salary * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = 1.05

emp1 = Employee("Bala", "Muthu", 30000)
emp2 = Employee("Ramesh", "Raghul", 50000)

print("Par Raise : 1")
print("**********")
print(Employee.raise_amt)
print(emp1.raise_amt)
print(emp2.raise_amt)

Employee.set_raise_amt(1.05)
print("Par Raise : 2")
print("**********")
print(Employee.raise_amt)
print(emp1.raise_amt)
print(emp2.raise_amt)
```

```
Par Raise : 1
**********
1.04
1.04
1.04
Par Raise : 2
**********
1.05
```

1.05
1.05

Class method as alternate constructors

```
class Employee:
    raise_amt = 1.04
    no_of_employees = 0

    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary
        self.email = self.fname + '.' + self.lname + '@gmail.com'
        Employee.no_of_employees += 1

    def fullname(self):
        return f"{self.fname} {self.lname}"

    def pay_raise(self):
        self.salary = int(self.salary * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = 1.05

    @classmethod
    def emp_details(cls, employee_details):
        fnm, lnm, sal = employee_details.split('-')
        return cls(fnm, lnm, sal)

emp1 = Employee.emp_details("Bala-Muthu-30000")
emp2 = Employee.emp_details("Ramesh-Raghul-50000")

print(emp1.email)
```

The class method “emp_details” will return

Employee(“Bala”,”Muthu”,30000) before instantiating the class object “emp1”

Static methods:

```
class Employee:
    raise_amt = 1.04
    no_of_employees = 0

    def __init__(self, fname, lname, salary):
        self.fname = fname
        self.lname = lname
        self.salary = salary
        self.email = self.fname + '.' + self.lname + '@gmail.com'
        Employee.no_of_employees += 1

    def fullname(self):
        return f"{self.fname} {self.lname}"

    def pay_raise(self):
        self.salary = int(self.salary * self.raise_amt)

    @classmethod
    def set_raise_amt(cls, amount):
        cls.raise_amt = 1.05

    @classmethod
    def emp_details(cls, employee_details):
        fnm, lnm, sal = employee_details.split('-')
        return cls(fnm, lnm, sal)

    @staticmethod
    def is_workday(day):
        if day.weekday() == 5 or day.weekday() == 6:
            return False
        return True


emp1 = Employee.emp_details("Bala-Muthu-30000")
emp2 = Employee.emp_details("Ramesh-Raghul-50000")

import datetime
my_date = datetime.date(2016,7,15)
print(emp1.is_workday(my_date))

my_date = datetime.date(2016,7,10)
print(Employee.is_workday(my_date))
```

The static methods, should have decorator '@staticmethod" before the method definition

It can be invoked by both class and objects

Output:

True
False