Object Oriented Programming-Python

Class

A class is a blueprint for the object.

We can think of class as a sketch of a vehicle with labels. It contains all the details about the regd_no, model, base_price, owner etc. Based on these descriptions, we can study about the vehicle.

```
class Vehicle():
      pass
```

Object

An object (instance) is an instantiation of a class. When class is defined, only the description for the object is defined. Therefore, no memory or storage is allocated.

```
benz=Vehicle()
```

Class and Instance Variables (Or attributes)

instance variables are variables whose value is assigned inside a constructor or method with self.

Class variables are variables whose value is assigned in class.

```
# Class for Computer Science Student
class Student:
     # Class Variable
    stream = 'cse'

    # The init method or constructor
    def __init__(self, roll):
        # Instance Variable
        self.roll = roll

# Objects of Student class
a = Student(101)
b = Student(102)

print(a.stream)  # prints "cse"
print(b.stream)  # prints "cse"
print(a.roll)    # prints 101

# Class variables can be accessed using class
# name also
print(Student.stream) # prints "cse"
```

We can define instance variables inside normal methods also.

```
# Class for EEE Student
class Student:
    # Class Variable
    stream = 'EEE'

    # The init method or constructor
    def __init__(self, roll):
        # Instance Variable
        self.roll = roll

    # Adds an instance variable inside normal method
    def setAddress(self, address):
        self.address = address

    # Retrieves instance variable
    def getAddress(self):
        return self.address

# Driver Code
a = Student(101)
a.setAddress("Bglr, Karnataka")
print(a.getAddress())
```

Class methods

The **@classmethod** decorator, is a built-in function decorator that is an expression that gets evaluated after your function is defined. The result of that evaluation shadows your function definition.
A class method receives the class as implicit first argument, just like an instance method receives the instance

```
class C(object):
    @classmethod
    def fun(cls, arg1, arg2, ...):
        ....
fun: function that needs to be converted into a class method
returns: a class method for function.
```

- A class method is a method which is bound to the class and not the object of the class.
- They have the access to the state of the class as it takes a class parameter that points to the class and not the object instance.
- It can modify a class state that would apply across all the instances of the class. For example, it can modify a class variable that will be applicable to all the instances.

Static methods

A static method does not receive an implicit first argument. The **@staticmethod** decorator, is a built-in function decorator that is an expression that gets evaluated after your function is defined.

```
class C(object):
    @staticmethod
    def fun(arg1, arg2, ...):
        ...
returns: a static method for function fun.
```

- A static method is also a method which is bound to the class and not the object of the class.
- A static method can't access or modify class state.

```
class Demo_Static_class_methods():
      def __init__(self):
            print "this is my constructor"

      def method(self):
            print "inside normal object method."

      @staticmethod
      def static_method(a, b):
            print "inside static method"
            return a+b

      @classmethod
      def class_method(cls, x, y,z):
            print "inside class method"
            return x+y+z

demo_Obj = Demo_Static_class_methods()
demo_Obj.method()
demo_Obj.class_method()
print "calling with class itself. Not through object"
Demo_Static_class_methods.class_method()
print "static method calling through object"
addition1 = demo_Obj.static_method(10,15)
print addition1
addition2 = Demo_Static_class_methods.class_method(100,200,300)
print addition2
```

Class method vs Static Method

- A class method takes cls as first parameter while a static method needs no specific parameters.
- A class method can access or modify class state while a static method can't access or modify it.

- In general, static methods know nothing about class state. They are utility type methods that take some parameters and work upon those parameters. On the otherhand class methods must have class as parameter.
- We use @classmethod decorator in python to create a class method and we use @staticmethod decorator to create a static method in python.

When to use what?

- We generally use class method to create factory methods. Factory methods return class object (similar to a constructor) for different use cases.
- We generally use static methods to create utility functions.

Example of a class and its instantiation(usage)

```
class Vehicle():
	# Constructor
	def __init__(self, regd_no, model, base_price):
		self.regd_no = regd_no
		self.model = model
		self.base_price=base_price
		self.on_road_price = None

	# Method to display vehicle details.
	def disply_vehicle(self):
		print "Vehicle Number: ", self.regd_no
		print "Model :", self.model
		print "Base Price: ", self.base_price
		print "On Road Price :", self.on_road_price

	# Method to calculate_onroad_price
	def calculate_onroad_price(self):
		cgst = .09
		sgst = .09
		road_tax = .04
		on_road_price = (1+cgst+sgst+road_tax)*self.base_price
		self.on_road_price = on_road_price

if __name__ == "__main__":
	benz = Vehicle("KA51MA2345", "C-210", 4356000)
	bmw = Vehicle("TN23AZ9876", "B-110H", 3289750)
	benz.calculate_onroad_price()
	benz.disply_vehicle()
	bmw.disply_vehicle()
```

Inheritance

Inheritance is a way of creating new class for using details of existing class without modifying it. The newly formed class is a derived class (or child class). Similarly, the existing class is a base class (or parent class).

- First statement of a derived class constructor should be the calling of base class constructor.

Multiple Inheritance in Python

A class can be derived from more than one base classes in Python. This is called multiple inheritance.

In multiple inheritance, the features of all the base classes are inherited into the derived class. The syntax for multiple inheritance is similar to single inheritance.

Example

```
class Base1:
    pass

class Base2:
    pass

class MultiDerived(Base1, Base2):
    pass
```

Multilevel Inheritance in Python

On the other hand, we can also inherit form a derived class. This is called multilevel inheritance. It can be of any depth in Python.

In multilevel inheritance, features of the base class and the derived class is inherited into the new derived class.

```
class Base:
    pass

class Derived1(Base):
    pass

class Derived2(Derived1):
    pass
```

Here, `Derived1` is derived from `Base`, and `Derived2` is derived from `Derived1`

Polymorphism

Polymorphism is an ability (in OOP) to use common interface for multiple form (data types).

Suppose, we need to calculate on-road price of vehicles, there are multiple vehicle option (bike, car, heavy vehicle). However, we could use same method to calculate the on-road price. This concept is called Polymorphism.

Example with Inheritance & Polymorphism

Below program explains the inheritance & polymorphism.

```
class Person():
      def __init__(self, first_name, last_name, age):
            self.first_name = first_name
            self.last_name = last_name
            self.age = age

      def display(self):
            print "First Name: ", self.first_name
            print "Last name :", self.last_name
            print "Age: ", self.age

class Student(Person):
      def __init__(self, first_name, last_name, age, course, marks=[]):
            Person.__init__(self, first_name, last_name, age)
            self.course = course
            self.marks = marks
            self.average = None

      def calcualte_average(self):
            if self.marks:
                  sum = reduce((lambda a,b: a+b),self.marks)
                  self.average = float(sum)/len(self.marks)
            else:
                  self.average = 0

      def display(self):
            print "First Name: ", self.first_name
            print "Last name :", self.last_name
            print "Age: ", self.age
            print "Marks :", self.marks
            print "Average: ", self.average

class Teaching_Faculty(Person):
      hra = 0.4
      ta = 0.2
      da = 1800
      pf = 0.12
      def __init__(self, first_name, last_name, age, subject, grade,
base_salary):
            Person.__init__(self, first_name, last_name, age)
```

```
            self.subject = subject
            self.grade = grade
            self.base_salary = base_salary

      def calcualte_net_salary(self):
            self.net_salary = self.gross_salary-(self.base_salary*self.pf)

      def calculate_gross_salary(self):
            self.gross_salary = (1+self.hra+self.ta+self.pf)*self.base_salary
+ self.da

      def display(self):
            print "First Name: ", self.first_name
            print "Last name :", self.last_name
            print "Age: ", self.age
            print "Subject: ", self.subject
            print "Grade: ", self.grade
            print "Net salary :", self.net_salary
            print "Gross Salary: ", self.gross_salary


class Administrative_Faculty(Person):
      hra = 0.3
      ta = 0.18
      da = 1450
      pf = 0.12
      def __init__(self, first_name, last_name, age, dept, base_salary):
            Person.__init__(self, first_name, last_name, age)
            self.dept = dept
            self.base_salary = base_salary

      def calcualte_net_salary(self):
            self.net_salary = self.gross_salary-(self.base_salary*self.pf)

      def calculate_gross_salary(self):
            self.gross_salary = (1+self.hra+self.ta+self.pf)*self.base_salary
+ self.da

      def display(self):
            print "First Name: ", self.first_name
            print "Last name :", self.last_name
            print "Age: ", self.age
            print "Dept: ", self.dept
            print "Basic Salry: ", self.base_salary
            print "Net salary :", self.net_salary
            print "Gross Salary: ", self.gross_salary


if __name__=="__main__":

      std1 = Student("Rajesh", "Sharma", 21, "ECE", [10,9,9,7,8,8])
      std1.calcualte_average()
      std1.display()

      science_faculty1 = Teaching_Faculty("Mahuri","Kumari", 42, "Science",
"Asst. Prof", 32000)
      science_faculty1.calculate_gross_salary()
```

```
        science_faculty1.calcualte_net_salary()
        science_faculty1.display()

        accountant_faculty1 = Administrative_Faculty("Satya","Swamy", 39,
"Accountant", 32000)
        accountant_faculty1.calculate_gross_salary()
        accountant_faculty1.calcualte_net_salary()
        accountant_faculty1.display()

        std2 = Student("Rajesh", "Sharma", 21, "ECE")
        std2.calcualte_average()
        std2.display()
```