

Tutorial_Session6

September 28, 2018

1 CLASSES

- A class is a template that provides a logical grouping of data and methods that operate on them.
- Instances of a class are called objects.
- Data and methods associated with a class are collectively known as class attributes.

1.1 Classes and Objects

- Variables used so far took values of types (also called classes) string (str), integer (int), floating point (float), Boolean (bool), list, tuple, or dictionary (dict).

```
In [ ]: print(type(12), type(12.5), type('hello'))
```

**** Accessing attribute of an instance of class**** * To specify an attribute of a class (or class instance), we write the name of the class (or class instance) followed by a dot, followed by the name of that attribute. The method **lower** defined in class **str** has been invoked for the object **name**.

```
In [ ]: name = 'Raman'
        lname = name.lower()
        print(lname)
```

Alternative way of invoking the method associated with an instance of class: * Specify the name of the class (str), followed by the dot operator (.), followed by the name of the method (lower), followed by an object (name). The object name being an argument is enclosed in parentheses.

```
In [ ]: lname = str.lower(name)
        print(lname)
```

1.2 PERSON class

1.2.1 Syntax of Class Definition

A class definition begins with the keyword **class** followed by the name of the class, and a colon. By convention, the first letter of the class name is capitalized. The syntax for class definition is as follows:

```
class ClassName:
    classBody
```

1.2.2 Operations supported by classes:

1. **Instantiation:** It refers to the creation of an object, i.e. an instance of the class.
2. **Attribute references:** Methods and data members of an object of a class are accessed using the notation: name of the object, followed by dot operator, followed by the member name.

```
In [ ]: class Person:
        ''' The class Person describes a person'''
        count = 0
        def __init__(self, name, DOB, address):#constructor
            '''
            Objective: To initialize object of class Person
            Input Parameters:
                self (implicit parameter) - object of type Person
                name - string
                DOB - string (Date of Birth)
                address - string
            Return Value: None
            '''
            self.name = name
            self.DOB = DOB
            self.address = address
            Person.count += 1

        def getName(self):
            '''
            Objective: To retrieve name of the person
            Input Parameter: self (implicit parameter) - object of type Person
            Return Value: name - string
            '''
            return self.name

        def getDOB(self):
            '''
            Objective: To retrieve the date of birth of a person
            Input Parameter: self (implicit parameter) - object of type Person
            Return Value: DOB - string
            '''
            return self.DOB

        def getAddress(self):
            '''
            Objective: To retrieve address of person
```

```

        Input Parameter: self (implicit parameter) - object of type Person
        Return Value: address - string
        '''
        return self.address

def getCount(self):
    '''
    Objective: To get count of objects of type Person
    Input Parameter: self (implicit parameter) - object of type Person
    Return Value: count: numeric
    '''
    return Person.count

def __str__(self): #converting to string
    '''
    Objective: To return string representation of object of type Person
    Input Parameter: self (implicit parameter)- object of type
    Person
    Return Value: string
    '''
    return 'Name: '+self.name+'\nDOB: '+str(self.DOB)\
    +'\nAddress: '+self.address

```

1.2.3 Creating an instance of class Person

```

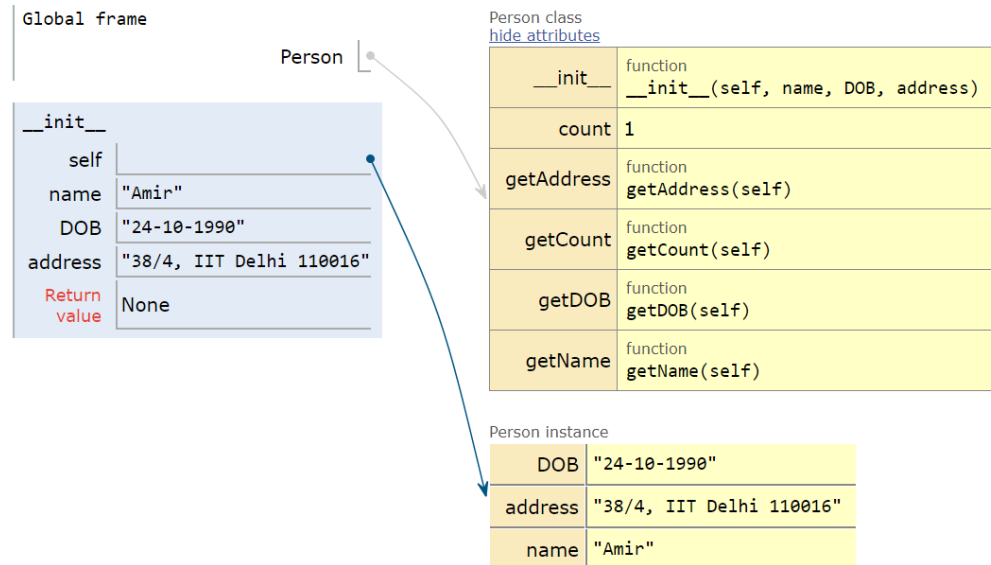
In [ ]: p1 = Person('Amir', '24-10-1990', '38/4, IIT Delhi 110016')
        print(Person.count)

```

The execution of the above statement does three things: 1. Creates an instance of class Person
 2. Initializes it by invoking the method `__init__` defined in lines 3. Returns a reference to it, so the name p1 now refers to the instance of the class Person that has just been created

By default, Python passes object itself (such as p1) as the first argument to the method `__init__`.

1.2.4 Instance p1 of class Person



1.2.5 Printing an instance of class

Python invokes `__str__` method of the corresponding class to obtain a string representation of the object

```
In [ ]: print(7)
        # OR
        print(int.__str__(7))
        print(str(7))

In [ ]: print(p1)
        print('*****')
        print(p1.__str__())
        print('*****')
        print(Person.__str__(p1))
        print('*****')
        print(str(p1))
```

```
In [ ]: p1.getDOB()
```

1.2.6 List of attributes of the object

```
In [ ]: dir(p1)
```

1.2.7 Deleting an object of class Person

```
In [ ]: def __del__(self):
        """
        Objective: To be invoked on deletion of an instance of the
        class Person
```

```

    Input Parameter:
    self (implicit parameter) object of type Person
    Return Value: None
    '''
    print('Deleted !!!')
    Person.count -= 1

```

```
In [ ]: p1.__del__ = __del__
```

```
In [ ]: del p1
```

```
In [ ]: print(p1)
```

1.3 Polymorphism

- applying a method/operator to objects of different types.
 - eg print method
-

1.4 Operator Overloading

- done with the help of function **add/sub/mul/_lt_** etc.

```

In [ ]: class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def __str__(self):
        return str((self.x, self.y))

    def __add__(self, other):
        x=self.x+other.x
        y=self.y+other.y
        return Point(x,y)

p1 = Point(2,3)
p2 = Point(3,4)
print(p1)
print(p2)
p3 = p1+p2

print('Sum is',p3)

```

1.5 Function Overloading

```
In [ ]: def area(radius):
        #Compute area of circle
        print("Inside circle area")
        areaCirc = 3.14*radius*radius
        return areaCirc

def area(length, breadth):
    #compute area of rectangle
    print("Inside rect area")
    areaRect = length*breadth
    return areaRect

a = area(4,6)
b = area(3)

In [ ]: def area(a,b=None):
        if (b==None):
            print("Inside circle area")
            areaCirc = 3.14*a*a
            return areaCirc
        else:
            print("Inside rect area")
            areaRect = a*b
            return areaRect

a = area(5)
b = area(5,6)
```

1.6 Data Hiding

```
In [ ]: class Point:
        def __init__(self, x, y):
            self.__x = x
            self.y = y

        def __str__(self):
            return str((self.x, self.y))

        def __add__(self, other):
            x=self.x+other.x
            y=self.y+other.y
            return Point(x,y)

p1 = Point(2,3)
```

```

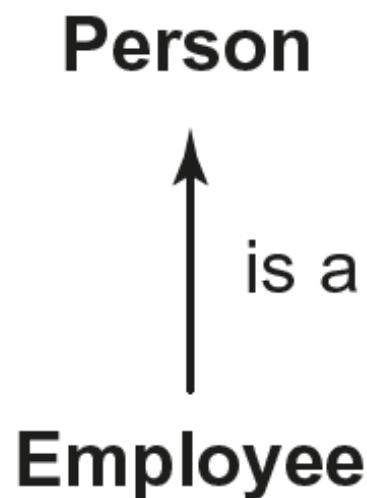
p2 = Point(3,4)
print(p1.x)
print(p2.y)
p3 = p1+p2

print('Sum is',p3)

```

1.7 Inheritance

- Inheritance is an important feature of object oriented programming that imparts ability to a class to inherit properties and behavior of another class



- In the language of Object-oriented Programming (OOP), we say that Employee class inherits or derives the data and method attributes from the Person class.
- Here, Person class is called base, super, or parent class, and Employee class is called derived, sub, or child class.

1.7.1 Single Inheritance

- When inheritance involves a derived class that derives its properties from a single base class, it is called **single inheritance**

```

In [ ]: class Employee(Person):
        nextId = 1001
        empCount = 0

        def __init__(self, name, DOB, address, basicSalary, dateOfJoining):
            '''
            Objective: To initialize an object of class Employee
            Input Parameters:
                self (implicit parameter)  object of type Employee

```

```

        name - string, address string
        DOB - Date of Birth object of type MyDate
        basicSalary - numeric value
        dateOfJoining object of type MyDate
    Return Value: None
    '''

    Person.__init__(self, name, DOB, address)
    self.idNum = Employee.nextId
    self.basicSalary = basicSalary
    self.dateOfJoining = dateOfJoining
    Employee.nextId += 1
    Employee.empCount += 1

def getId(self):
    '''
    Objective: To retrieve id of the Employee
    Input Parameter: self (implicit parameter) object of type Employee
    Return Value: id - numeric value
    '''
    return self.idNum

def getSalary(self):
    '''
    Objective: To retrieve salary of the Employee
    Input Parameter: self (implicit parameter) - object of type Employee
    Return Value: basicSalary - numeric value
    '''
    return self.basicSalary

def reviseSalary(self, newSalary):
    '''
    Objective: To update salary of the Employee
    Input Parameters: self (implicit parameter) - object of type Employee
    newSalary - numeric value
    Return Value: None
    '''
    self.basicSalary = newSalary

def getJoiningDate(self):
    '''
    Objective: To retrieve joining date of the Employee
    Input Parameter: self (implicit parameter) - object of type Employee
    Return Value: dateOfJoining - object of type MyDate
    '''
    return self.dateOfJoining

def __str__(self):
    '''

```


Objective: To return string representation of object of type Employee.
Input Parameter: self (implicit parameter) - object of type Employee
Return Value: string
 '''

```
return Person.__str__(self)+'\nId:'+str(self.getId())+\
'\nSalary:'+str(self.getSalary())+\
'\nDate of Joining:'+str(self.getJoiningDate())
```

```
In [ ]: emp1 = Employee('Rehman', '5 June 1990', ' D-9, Vivek Vihar, Delhi', 50000, '2 August 2018')
```

```
In [ ]: print(Employee.empCount)
```

- Call to the method **init** is made using a superclass name and the object instance is explicitly passed as an argument to the superclass method.
- Alternatively, we may use the **super** function to access a method of the superclass.

```
super(Employee, self).__init__(name, DOB, address)
super().__init__(name, DOB, address)
```

1.8 5. Built-in Functions for Classes

1.8.1 Function `issubclass`

- The function `issubclass` returns True if sub is the subclass of class super, and False otherwise.

```
issubclass(sub, super)
```

```
In [ ]: issubclass(Employee, Person)
```

1.8.2 Function `isinstance`

- The function `isinstance` returns True if either obj is an instance of class class1 or it is an instance of a subclass of class class1.

```
isinstance(obj, class1)
```

```
In [ ]: isinstance(emp1, Person)
```

1.8.3 Function `hasattr`

- The function `hasattr` returns True if instance obj contains an attribute attr, and False otherwise.

```
hasattr(obj, attr)
```

```
In [ ]: hasattr(emp1, 'dateOfJoining')
```

1.9 Multilevel/Multiple Inheritance

- Python supports Multilevel Inheritance
 - class Employee(Person):
 - class DailyWager(Employee):
 - class A(B,C):
-

1.10 Assignment

Define a base class Vehicle, having attributes registration number, make, model and color. Also define classes PassengerVehicle and CommercialVehicle that are derived from the class Vehicle. The PassengerVehicle class should have additional attribute for maximum passenger capacity. The CommercialVehicle class should have an additional attribute for maximum load capacity. Define **init** method for all these classes. Also define get and set methods to retrieve and set the value of the data attributes.