Lecture-34

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.

The example for object of parrot class can be:

```
obj = Parrot()
```

Here, obj is an object of class Parrot.

Suppose we have details of parrots. Now, we are going to show how to build the class and objects of parrots.

Instantiating objects:

To instantiate an object, type the class name, followed by two brackets. You can assign this to a variable to keep track of the object.

ozzy = Dog() And print it:

print(ozzy)<__main__.Dog object at 0x111f47278>

Adding attributes to a class:

After printing ozzy, it is clear that this object is a dog. But you haven't added any attributes yet. Let's give the Dog class a name and age, by rewriting it:

```
class Dog:
        def __init__(self, name, age):
            self.name = name
            self.age = age
```

You can see that the function now takes two arguments after self: name and age. These then get assigned to self.name and self.age respectively. You can now create a new ozzy object, with a name and age:

```
ozzy = Dog("Tommy", 2)
```

To access an object's attributes in Python, you can use the dot notation. This is done by typing the name of the object, followed by a dot and the attribute's name.

```
class Dog:
def __init__(self, name, age):
        self.name = name
        self.age = age
ozzy = Dog("Tommy", 2)
print('Dog name:',ozzy.name)
print('Do
```

OUTPUT:

```
   Dog name: Tommy
Dog age: 2 g age:',ozzy.age)
```

• Define methods in a class:

Now that you have aDog class, it does have a name and age which you can keep track of, but it doesn't actually do anything. This is where instance methods come in. You can rewrite the class to now include a bark() method. Notice how the def keyword is used again, as well as the self argument.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def bark(self):
        print("bark bark!")
```

OUTPUT:

```
>>>
```

The bark method can now be called using the dot notation, after instantiating a new ozzy object. The method should print "bark bark!" to the screen. Notice the parentheses (curly brackets) in .bark(). These are always used when calling a method. They're empty in this case, since the bark() method does not take any arguments.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def bark(self):
        print("bark bark!")
ozzy = Dog("Ozzy", 2)
ozzy.bark()
```

OUTPUT:

```
bark bark!
```

Recall how you printed ozzy earlier? The code below now implements this functionality in the Dog class, with the doginfo() method. You then instantiate some objects with different properties, and call the method on them.

```
class Dog:
    def __init__(self, name, age):
        self.name = name
        self.age = age
def doginfo(self):
        print(self.name + " is " + str(self.age) + " year(s) old.")
ozzy = Dog("Ozzy", 2)
skippy = Dog("Skippy", 12)
filou = Dog("Filou", 8)
ozzy.doginfo()
skippy.doginfo()
filou.doginfo()
```

OUTPUT:

```
Ozzy is 2 year(s) old.

Skippy is 12 year(s) old.

Filou is 8 year(s) old.
```

As you can see, you can call the doginfo() method on objects with the dot notation. The response now depends on which Dog object you are calling the method on.

The __init__() Function

All classes have a function called __init__(), which is always executed when the class is being initiated.

Use the __init__() function to assign values to object properties, or other operations that are necessary to do when the object is being created:

Example:

Create a class named Person, use the __init__() function to assign values for name and age:

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

p1 = Person("John", 36)

print(p1.name)

print(p1.age)
```

Output:

```
John
36
```

Creating Instance Objects:

Example: Python Class

```
class Employee:
'Common base class for all employees'
    empCount = 0
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1
     def displayEmployee(self):
print "Name : ", self.name,  ", Salary: ", self.salary “
```

```
# “This would create first object of Employee class“
emp1 = Employee("Zara", 2000)
#"This would create second object of Employee class“
emp2 = Employee("Manni“,5000)
emp1.displayEmployee()
emp2.displayEmployee()
```

OUTPUT:

```
Name :  Zara , Salary:  2000
Name :  Manni , Salary:  5000
```

Accessing Attributes:

You access the object's attributes using the dot operator with object. Class variable would be accessed using class name as follows –

```
emp1.displayEmployee()

emp2.displayEmployee()

print "Total Employee %d" % Employee.empCount
```

Now, putting all the concepts together –

```
class Employee:
 'Common base class for all employees'
     empCount = 0
     def __init__(self, name, salary):
          self.name = name
          self.salary = salary
          Employee.empCount += 1
   def displayEmployee(self):
        print ("Name : ", self.name,  ", Salary: ", self.salary)
 #"This would create first object of Employee class“
emp1 = Employee("Zara", 2000)
```

```
#"This would create second object of Employee class“
emp2 = Employee("Manni“,5000)
emp1.displayEmployee()
emp2.displayEmployee()
print ("Total Employee %d" % Employee.empCount)
```

When the above code is executed, it produces the following result –

```
Name :  Zara ,Salary:  2000
Name :  Manni ,Salary:  5000
Total Employee 2
```

Accessing Attributes and Methods in Python:

Attributes of a class are function objects that define corresponding methods of its instances. They are used to implement access controls of the classes. Attributes of a class can also be accessed using the following built-in methods and functions :

The **getattr(obj, name[, default])** – to access the attribute of object.

The **hasattr(obj,name)** – to check if an attribute exists or not.

The **setattr(obj,name,value)** – to set an attribute. If attribute does not exist, then it would be created.

The **delattr(obj, name)** – to delete an attribute.

Example: # Python code for accessing attributes of class

```
class emp:
    name='Harsh'
    salary='25000'
    def show(self):
        print (self.name)
        print (self.salary )
e1 = emp()
# Use getattr instead of e1.name
print (getattr(e1,'name'))
  # returns true if object has attribute
print (hasattr(e1,'name'))
```

```
# sets an  attribute
setattr(e1,'height',152)
  # returns the value of attribute name height
print (getattr(e1,'height') )
  # delete the attribute
delattr(emp,'salary')
# returns true if object has attribute
print (hasattr(e1,' salary '))
```

Output :

```
Harsh
True
152
False
```

References:

1.Introduction to Computation and Programming using Python, by John Guttag, PHI Publisher

2. Fundamentals of Python first Programmes by Kenneth A Lambert, Copyrighted material Course Technology Inc. 1 st edition (6th February 2009)

3. https://www.tutorialspoint.com/python/index.htm

4. https://www.geeksforgeeks.org/python-programming-language

5. https://www.w3schools.com/python/

****END OF THE LECTURE***

THANK YOU