PYTHON OOPs

<u>**Attributes**</u> **:** Class attributes are the attributes which are owned by the class itself.

· Attributes will be shared by all the instances of the class. So, they have the same value for every instance.

· Class Attribute are placed outside of all methods in class . Generally they are placed immediately after class header.

Example :

```
class Support:
    A = "python supports OOPs concepts"
x = Support()
print (x.A)
```

Output :

```
python supports OOPs concepts
```

· In example we can say that here Support is a class with class attribute as A which is instantiated to an object x.

<u>**Accessing Attributes**</u> **:** After declaring the attributes to access these attributes in the class we can use **dot operator(.)** along with the object to which the class is instantiated.

· To access the class attributes we have to use class names.

Example :

```
class Student:
    'First class atrribute'
    stdCount = 0
    def __init__(self,name,department):
        self.name = name
        self.department = department
        Student.stdCount += 1
```

```
def studentdetails(self):
    print ("Name:",self.name,"department:",self.department)
std1=Student("Rajesh","Electrical")
std2 = Student("Shekar","Civil")
print (std1.studentdetails())
print (std1.name)
print (std2.department)
print (Student.stdCount)
```

Output :

```
Name: Rajesh department: Electrical
Rajesh
Civil
2
```

· In the example we declared a class attribute as stdCount and methods which contains attributes.

· Method attributes can be accessed with objname.attributename.

· Class attributes are accessed using classname.attributename.

Functions for Class Attributes :

· **setattr(obj,name,value)** help to set an attribute. If it does not exist it will create a attribute with the name specified.

· **getattr(obj,name[,default])** is used to access a attribute in the object.

· **hasattr(obj,name)** check whether the attribute with the specified name exist in object or not.

· **delattr(obj,name)** function will delete the attribute from the object.

Example:

```
class Student:
```

```
                    def __init__(self,name,department):
                        self.name = name
                        self.department = department
                    def studentdetails(self):
                    print
("Name:",self.name,"department:",self.department)
                std1=Student("Rajesh","Electrical")
                print (hasattr(std1,'name'))
                setattr(std1,'department','Civil')
                print (getattr(std1,'department'))
                delattr(std1,'name')
                print (getattr(std1,'department'))
                print (getattr(std1,'name'))
```

Output :

```
True
Civil
Civil
Traceback (most recent call last):
    File "attributes.py", line 17, in <module>
        print (std1.studentdetails())
File "attributes.py", line 9, in studentdetails
print ("Name:",self.name,"department:",self.department)
AttributeError: 'Student' object has no attribute 'name'
```

· In example we used attribute function and accessed and changed value of the attributes.

· Change the value of department to civil by using setattr and checked department attribute exist or not with hasattr and deleted the name attribute using delattr.

· Here it shows error when tried to get name attribute value because we have already deleted the name attribute form the object.

Built-in functions : Python has some built in function that are used to get information about the attributes.

· These built-in function names are prefixed and suffixed using underscore.

· **__dict__**: It will display a dictionary containing all attributes along with its values of the class.

. **__doc__**: It will print the documentation of the class if exist and None if not defined.

· **__name__**: It will give the name of the class.

· **__module__**: It will return the module name in which it is defined otherwise it will return “__main__”.

· **__bases__**: This is used in inheritance when we use parent and child class. It will check where the class is base class are a child of another class and return the base class name.

Example:

```
class Student:
    'First class atrribute’
    def __init__(self,name,department):
        self.name = name
        self.department = department
    def studentdetails(self):
        print
("Name:",self.name,"department:",self.department)
print (Student.__doc__)
print (Student.__name__)
print (Student.__module__)
print (Student.__bases__)
print (Student.__dict__)
```

Output :

```
First class atrribute

Student

__main__

(<class 'object'>,)

{'__module__': '__main__', '__doc__': 'First class atrribute', '__init__': <function Student.__init__ at 0x0000025EDAD0F620>, 'studentdetails': <function Student.studentdetails at 0x0000025EDAD0F6A8>, '__dict__': <attribute '__dict__' of 'Student' objects>, '__weakref__': <attribute '__weakref__' of 'Student' objects>}.
```