A class is the basic building block in python. A class creates a new type and object is the instance of the class. Classes provide a blueprint or a template using which objects are created.

**Class :** A class may contains data in the form of fields. Fields are also called attributes or class members and coded in the form of procedure known as method or class methods.

**Syntax:**

class class\_name:

<statement-1>

<statement-2>

<statement-n>

It starts with a keyword class followed by the class\_name and a colon (:). Variables defined in a class are called class variables and functions defined inside a class are called class methods. Class variables and class methods are together known as class members. The class members can be accessed through class objects. Class methods have access to all the data contained in the instance of the object.

**Creating objects**

Once a class is defined, the next job is to create an object of that class. The object can then access class variables and class methods using the dot operator(.). The syntax to create an object is given as,

**object\_name=class\_name()**

Creating an object or instance of a class is known as class instantiation.

**object\_name.class\_member\_name**

**Example:**

class abc:

var=10

x=abc()

print(x.var)

**CLASS METHOD AND SELF ARGUMENT**

Class methods are exactly same as ordinary functions that we have been defining so far with just one small difference. Class methods must have the first argument named as self. This is the first argument that is added to the beginning of the parameter list. Moreover, we do not pass a value for this parameter when we call the method. Python provides its value automatically. The self arguments refer to the object itself. That is, the object that has called the method. This means that even if a method that takes no arguments, it should be defined to accept the self. Similarly, a function defined to accept one parameter will actually take two-self and the parameter, so on and so forth.

Since, the class methods uses self, they require an object or instance of the class to be used. For this reason, they are often referred to as instance methods.

**Example:**

class abc:

var=10

def disp(self):

print(" class method ")

x=abc()

print(x.var)

x.disp()

**THE \_\_init\_\_() METHOD ( THE CLASS CONSTRUCTOR)**

The \_\_init\_\_() method has a special significance in python classes. The \_\_init\_\_() method is automatically executed when an object of a class is created. The method is useful to initialize the variables of the class object. Note the \_\_init\_\_() is prefixed as well as suffixed by double underscores. The \_\_init\_\_() method can be declared as, def \_\_init\_\_(self, [args…]).

**Example:**

class abc():

def \_\_init\_\_(self,val):

print(" class method ")

self.val=val

print(" the value is ",val)

obj=abc(22)

**CLASS VARIABLES AND OBJECT VARIABLES**

Basically, variables are of two types- class variables and object variables. As the name suggests, class variables are owned by the class and object variables are owned by each object. What this specifically means can be understood by using the following points.

1. If a class has n objects, then there will be n separate copies of the object variable each object will have its own object variable.
2. The object variable is not shared between objects.
3. A change made to the object variable by one object will not be reflected in other objects.
4. If a class has one class variable, then there will be one copy only for that variable. All the objects of that class will share the class variable.
5. Since there exists a single copy of the class variable, any change made to the class variable by an object will be reflected in all other objects.

**Example:**

**Output:**

The object value is: 10

The value of class variable is 1

The object value is: 20

The value of class variable is 2

The object value is: 30

The value of class variable is 3

class abc():

class\_var=0

def \_\_init\_\_(self,var):

abc.class\_var+=1

self.var=var

print(" The object value is:",var)

print(" The value of class variable is",abc.class\_var)

obj1=abc(10)

obj2=abc(20)

obj3=abc(30)

**PUBLIC AND PRIVATE DATA MEMBERS**

Public variables are those variables that are defined in the class and can be accessed from anywhere in the program, of course using the dot operator. Here, anywhere from the program means that the public variables can be accessed from within the class as well as from outside the class in which it is defined.

Private variables, on the other hand, are those variables that are defined in the class with a double score prefix(\_\_). These variables can be accessed only from within the class and from nowhere outside the class.

**Output:**

From class method, var1 30

From class method, var2 23

From main module,var1 30

Traceback (most recent call last):

File "C:/Users/admin/Desktop/pythonEx/class.py", line 11, in <module>

print("From main module,var2",obj.\_\_var2)

AttributeError: 'abc' object has no attribute '\_\_var2'

**Example:**

class abc():

def \_\_init\_\_(self,var1,var2):

self.var1=var1

self.\_\_var2=var2

def disp(self):

print(" From class method, var1",self.var1)

print(" From class method, var2",self.\_\_var2)

obj=abc(30,23)

obj.disp()

print("From main module,var1",obj.var1)

print("From main module,var2",obj.\_\_var2)

As a good programming habit, we should never try to access a private variable from anywhere outside the class, but if for some reason, we need to do it, then we can access the private variable using the following syntax:

**Object\_name.\_classname\_\_privatevariable**

So, to remove the error from the above code, we could have written the last statement as

**print(" From class method, var2",obj.\_abc\_\_var2)**

**PRIVATE METHODS**

Private members/attributes should not be accessed from anywhere outside the class. Like private attributes, we have private methods in our class. Like private attributes, we should not use private methods anywhere outside the class. If it is necessary to access the methods outside the class, then they are accessed with a small difference. A private method can be accessed using the object name as well as the class name from outside the class. The syntax for accessing the private method in such a case would be.

**Objectname.\_classname\_privatemethodname**

**Example:**

class abc():

**Output:**

from class method, var= 10

def \_\_init\_\_(self,var):

self.\_\_var=var

def \_\_disp(self):

print(" from class method, var=",self.\_\_var)

obj=abc(10)

obj.\_abc\_\_disp()

**BUILT-IN CLASS ATTRIBUTES**

Every class defined in Python has some built-in attributes associated with it. Like other attributes, these attributes can also be accessed using dot operator.

**.\_dict\_\_:** The attribute gives a dictionary containing the class’s or object’s namespace.

**.\_\_doc\_\_:** The attribute gives the class documentation string if specified. In case the documentation string is not specified, then the attribute return none.

**.\_\_name\_\_:** The attribute returns the name of the class.

**.\_\_module\_\_:** The attribute gives the name of the module in which the class is defined.

**.\_\_bases\_\_:** The attribute is used in inheritance to return the base classes in the order of their occurrence in the base class list.

**Example:**

class abc():

def \_\_init\_\_(self,var1,var2):

**Output:**

var1 is : 43

var2 is : 34

object.\_\_dict\_\_ {'var1': 43, 'var2': 34}

object.\_\_doc\_\_ None

class.\_\_name\_\_ abc

object.\_\_module\_\_ \_\_main\_\_

class.\_\_bases\_\_ (<class 'object'>,)

self.var1=var1

self.var2=var2

def disp(self):

print(" var1 is :",self.var1)

print(" var2 is :",self.var2)

obj=abc(43,34)

obj.disp()

print("object.\_\_dict\_\_ ",obj.\_\_dict\_\_)

print("object.\_\_doc\_\_ ",obj.\_\_doc\_\_)

print("class.\_\_name\_\_ ",abc.\_\_name\_\_)

print("object.\_\_module\_\_ ",obj.\_\_module\_\_)

print("class.\_\_bases\_\_ ",abc.\_\_bases\_\_)

**STATIC METHODS**

Static methods are special case of methods. Any functionality that belongs to a class, but that does not require the object is placed in the static method. Static methods are similar to class methods. The only difference is that a static method does not receive any additional arguments. They are just like normal functions that belong to a class.

**Syntax:** @staticmethod

Def name(args):

Statements

A static method can be called either on the clss or on an instance. When it is called with an instance, the instance is ignored except for its class.

class person:

def \_\_init\_\_(self,name,age):

self.name=name

self.age=age

@staticmethod

def isadult(age):

return age>18

res=person.isadult(12)

print("is person adult",res)

res=person.isadult(23)

print("\n is person adult",res)

**Example:**

class maths:

@staticmethod

def addnum(n1,n2):

return n1+n2

res=maths.addnum(23,2)

print("The result is ",res)