

## Python Class

Python is a completely object-oriented language. You have been working with classes and objects right from the beginning of these tutorials. Every element in a Python program is an object of a class. A number, string, list, dictionary, etc., used in a program is an object of a corresponding built-in class. You can retrieve the class name of variables or objects using the `type()` method, as shown below.

### Example: Python Built-in Classes

[Copy](#)

```
>>> num=20
>>> type(num)
<class 'int'>
>>> s="Python"
>>> type(s)
<class 'str'>
```

## Defining a Class

A class in Python can be defined using the `class` keyword.

```
class <ClassName>:
    <statement1>
    <statement2>
    .
    .
    <statementN>
```

As per the syntax above, a class is defined using the `class` keyword followed by the class name and `:` operator after the class name, which allows you to continue in the next indented line to define class members. The followings are class members.

1. [Class Attributes](#)
2. [Constructor](#)
3. [Instance Attributes](#)
4. [Properties](#)
5. [Class Methods](#)

A class can also be defined without any members. The following example defines an empty class using the `pass` keyword.

### Example: Define Python Class

[Copy](#)

```
class Student:
    pass
```

Class instantiation uses function notation. To create an object of the class, just call a class like a parameterless function that returns a new object of the class, as shown below.

### Example: Creating an Object of a Class

[Copy](#)

```
std = Student()
```

Above, `Student()` returns an object of the `Student` class, which is assigned to a local [variable](#) `std`. The `Student` class is an empty class because it does not contain any members.

### Class Attributes

Class attributes are the variables defined directly in the class that are shared by all objects of the class. Class attributes can be accessed using the class name as well as using the objects.

|   |      |
|---|------|
| Example: Define Python Class                            | Copy |
| <pre>class Student:     schoolName = 'XYZ School'</pre> |      |

Above, the `schoolName` is a class attribute defined inside a class. The value of the `schoolName` will remain the same for all the objects unless modified explicitly.

|   |      |
|---|------|
| Example: Define Python Class  | Copy |
| <pre>&gt;&gt;&gt; Student.schoolName 'XYZ School' &gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.schoolName 'XYZ School'</pre> |      |

As you can see, a class attribute is accessed by `Student.schoolName` as well as `std.schoolName`. Changing the value of class attribute using the class name would change it across all instances. However, changing class attribute value using instance will not reflect to other instances or class.

|  |      |
|--|------|
| Example: Define Python Class   | Copy |
| <pre>&gt;&gt;&gt; Student.schoolName = 'ABC School' # change attribute value using class name &gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.schoolName 'ABC School' # value changed for all instances &gt;&gt;&gt; std.schoolName = 'My School' # changing instance's attribute &gt;&gt;&gt; std.schoolName 'My School' &gt;&gt;&gt; Student.schoolName # instance level change not reflectd to class attribute 'ABC School' &gt;&gt;&gt; std2 = Student() &gt;&gt;&gt; std2.schoolName 'ABC School'</pre> |      |

The following example demonstrates the use of class attribute `count`.

|  |      |
|--|------|
| Example: Student.py  | Copy |
| <pre>class Student:     count = 0     def __init__(self):         Student.count += 1</pre> |      |

In the above example, `count` is an attribute in the `Student` class. Whenever a new object is created, the value of `count` is incremented by 1. You can now access the `count` attribute after creating the objects, as shown below.


Example:

Copy


```
>>> std1=Student()
>>> Student.count
1
>>> std2 = Student()
>>> Student.count
2
```

'Baby Shark' Becomes 1st YouTube Video To Surpass 10 B...


NOW PLAYING



A Look Ahead at Xbox's Plans for 2022



XBOX



Grand

**'Baby Shark'...**  
Baby Shark Becomes 1st YouTube Video , To Surpass 10... [Watch Video](#)

**A Look Ahead at...**  
A Look Ahead , at Xbox's Plans , for 2022.Kotaku reports that... [Watch Video](#)

**Microsoft Gaming...**  
Microsoft Gaming Executive , Says Netflix's Move Into... [Watch Video](#)

**GTA Publisher...**  
GTA Publisher

Constructor

In Python, the constructor method is invoked automatically whenever a new object of a class is instantiated, same as constructors in C# or Java. The constructor must have a special name `__init__()` and a special parameter called `self` .

Note:

The first parameter of each method in a class must be the `self` , which refers to the calling object. However, you can give any name to the first parameter, not necessarily `self` .

The following example defines a constructor.

Example: Constructor

Copy

```
class Student:
    def __init__(self): # constructor method
        print('Constructor invoked')
```

Now, whenever you create an object of the `student` class, the `__init__()` constructor method will be called, as shown below.

Example: Constructor Call on Creating Object

Copy


```
>>>s1 = Student()
Constructor invoked
>>>s2 = Student()
Constructor invoked
```

The constructor in Python is used to define the attributes of an instance and assign values to them.


Instance Attributes

Instance attributes are attributes or properties attached to an instance of a class. Instance attributes are defined in the constructor.


The following example defines instance attributes `name` and `age` in the constructor.

|  |  |
|--|--|
| Example: Instance Attributes   |  Copy |
| <pre>class Student:     schoolName = 'XYZ School' # class attribute      def __init__(self): # constructor         self.name = '' # instance attribute         self.age = 0 # instance attribute</pre> |  |


An instance attribute can be accessed using dot notation: `[instance name].[attribute name]`, as shown below.

|   |  |
|---|--|
| Example:  |  Copy |
| <pre>&gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.name '' &gt;&gt;&gt; std.age 0</pre> |  |


You can set the value of attributes using the dot notation, as shown below.

|  |  |
|--|--|
| Example:   |  Copy |
| <pre>&gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.name = "Bill" # assign value to instance attribute &gt;&gt;&gt; std.age=25        # assign value to instance attribute &gt;&gt;&gt; std.name          # access instance attribute value Bill &gt;&gt;&gt; std.age           # access value to instance attribute 25</pre> |  |

You can specify the values of instance attributes through the constructor. The following constructor includes the `name` and `age` parameters, other than the `self` parameter.

|  |  |
|--|--|
| Example: Setting Attribute Values  |  Copy |
| <pre>class Student:     def __init__(self, name, age):         self.name = name         self.age = age</pre> |  |

Now, you can specify the values while creating an instance, as shown below.

|   |  |
|---|--|
| Example: Passing Instance Attribute Values in Constructor   |  Copy |
| <pre>&gt;&gt;&gt; std = Student('Bill',25) &gt;&gt;&gt; std.name 'Bill' &gt;&gt;&gt; std.age 25</pre> |  |

#### Note:

You don't have to specify the value of the `self` parameter. It will be assigned internally in Python.

You can also set default values to the instance attributes. The following code sets the default values of the constructor parameters. So, if the values are not provided when creating an object, the values will be assigned latter.

#### Example: Setting Default Values of Attributes

[Copy](#)

```
class Student:
    def __init__(self, name="Guest", age=25)
        self.name=name
        self.age=age
```

Now, you can create an object with default values, as shown below.

#### Example: Instance Attribute Default Value

[Copy](#)

```
>>> std = Student()
>>> std.name
'Guest'
>>> std.age
25
```

Visit [class attributes vs instance attributes in Python](#) for more information.

## Class Properties

In Python, a property in the class can be defined using the [property\(\) function](#).

The `property()` method in Python provides an interface to instance attributes. It encapsulates instance attributes and provides a property, same as Java and C#.

The `property()` method takes the `get`, `set` and `delete` methods as arguments and returns an object of the `property` class.

The following example demonstrates how to create a property in Python using the `property()` function.


#### Example: property()

[Copy](#)

```
class Student:
    def __init__(self):
        self.__name=''
    def setname(self, name):
        print('setname() called')
        self.__name=name
    def getname(self):
        print('getname() called')
        return self.__name
    name=property(getname, setname)
```

In the above example, `property(getname, setname)` returns the property object and assigns it to `name`. Thus, the `name` property hides the [private instance attribute](#) `__name`. The `name` property is accessed directly, but internally it will invoke the `getname()` or `setname()` method, as shown


below.

|   |  |
|---|--|
| Example: property()   |  Copy |
| <pre>&gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.name="Steve" setname() called &gt;&gt;&gt; std.name getname() called 'Steve'</pre> |  |

It is recommended to use the [property decorator](#) instead of the `property()` method.


Class Methods

You can define as many methods as you want in a class using the `def` keyword. Each method must have the first parameter, generally named as `self` , which refers to the calling instance.


|  |   |
|--|---|
| Example: Class Method  |  Copy |
| <pre>class Student:     def displayInfo(self): # class method         print('Student Information')</pre> |   |

`self` is just a conventional name for the first argument of a method in the class. A method defined as `mymethod(self, a, b)` should be called as `x.mymethod(a, b)` for the object `x` of the class.

The above class method can be called as a normal function, as shown below.

|  |  |
|--|--|
| Example: Class Method  |  Copy |
| <pre>&gt;&gt;&gt; std = Student() &gt;&gt;&gt; std.displayInfo() 'Student Information'</pre> |  |

The first parameter of the method need not be named `self` . You can give any name that refers to the instance of the calling method. The following `displayInfo()` method names the first parameter as `obj` instead of `self` and that works perfectly fine.

|   |  |
|---|--|
| Example: Class Method   |  Copy |
| <pre>class Student:     def displayInfo(obj): # class method         print('Student Information')</pre> |  |

Defining a method in the class without the `self` parameter would raise an exception when calling a method.

|   |  |
|---|--|
| Example: Class Method   |  Copy |
| <pre>class Student:     def displayInfo(): # method without self parameter         print('Student Information')</pre> |  |

```
>>> std = Student()
>>> std.displayInfo()
Traceback (most recent call last):
  std.displayInfo()
TypeError: displayInfo() takes 0 positional arguments but 1 was given
```

The method can access instance attributes using the `self` parameter.

Example: Class Method

 Copy

```
class Student:
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def displayInfo(self): # class method
        print('Student Name: ', self.name, ', Age: ', self.age)
```

You can now invoke the method, as shown below.

Example: Calling a Method

 Copy

```
>>> std = Student('Steve', 25)
>>> std.displayInfo()
Student Name: Steve , Age: 25
```

## Deleting Attribute, Object, Class

You can delete attributes, objects, or the class itself, using the `del` keyword, as shown below.

Example: Delete Attribute, Object, Class

 Copy

```
>>> std = Student('Steve', 25)
>>> del std.name # deleting attribute
>>> std.name
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    std.name
AttributeError: 'Student' object has no attribute 'name'
>>> del std # deleting object
>>> std.name
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    std.name
NameError: name 'std' is not defined
>>> del Student # deleting class
>>> std = Student('Steve', 25)
Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    std = Student()
NameError: name 'Student' is not defined
```

Learn Python using coding questions with answers.

[Python Questions & Answers](#)

Want to check how much you know Python?

[Start Python Skill Test](#)