**self in Python class**

It represents the instance of the class being used. **self** is used as the first parameter in instance methods .

**Self in the Constructor (\_\_init\_\_ method)**

Allows the method to set instance attributes for the object being created

**Self in Instance Methods**

method within the class that operates on an instance of the class must include self as the first parameter

**Self: Pointer to the Current Object**

**class** **Check**:

**def** \_\_init\_\_(self):

print("Address of self = ", id(self))

obj = Check()

print("Address of class object = ", id(obj))

**Class and Instance Attributes in Python**

**Class attributes:**Class attributes belong to the class itself they will be shared by all the instances.

**class** **sampleclass**:

count = 0

**def** increase(self):

sampleclass.count += 1

s1 = sampleclass()

s1.increase()

print(s1.count)

**instance** **attributes** are not shared by objects. Every object has its own copy of the instance attribute

**class** **emp**:

**def** \_\_init\_\_(self):

self.name = 'xyz'

self.salary = 4000

**def** show(self):

e1 = emp()

print("Dictionary form :", vars(e1)) //Display attribute instance in

form of dictionary

print(dir(e1)) // ['\_\_doc\_\_', '\_\_init\_\_', '\_\_module\_\_', 'name', 'salary', 'show']

**Create a Python Subclass**

**subclass** is a class that inherits attributes and methods from another class

**Creating a simple subclass**

**class** **Animal**:

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

self.name = name

**class** **Dog**(Animal):

**def** sound(self):

**return** "Woof!"

a = Animal("Generic Animal")

d = Dog("Buddy")

print(a.name) *# Output: Generic Animal*

print(d.name) *# Output: Buddy*

print(d.sound()) *# Output: Woof!*

**Adding additional attributes in the subclass**

 It calls the superclass constructor using [super()](https://www.geeksforgeeks.org/python-super/) to initialize the common attribute.

**class** **Shape**:

**def** \_\_init\_\_(self, color):

self.color = color

**class** **Circle**(Shape):

**def** \_\_init\_\_(self, color, radius):

super().\_\_init\_\_(color)

self.radius = radius

**def** area(self):

**return** 3.14 \* self.radius \*\* 2

**Inner Class in Python**

A class defined in another class is known as an **inner class** or nested class.

To hide the inner class from the outside world.

**class** **Color**:

**def** \_\_init\_\_(self):

self.name = 'Green'

self.lg = self.Lightgreen() //Instance of inner class

**def** show(self):

print('Name:', self.name)

**class** **Lightgreen**:

**def** \_\_init\_\_(self):

self.name = 'Light Green'

self.code = '024avc'

**def** display(self):

print('Name:', self.name)

print('Code:', self.code)

outer = Color()

outer.show()

g = outer.lg

g.display()

Types of inner classes are as follows:

1. Multiple inner class

The class contains one or more inner classes

1. Multilevel inner class

The class contains an inner class and that inner class again contains another inner class

**Python MetaClasses ^ ^ : Half Done**

The classes that generate other classes are defined as metaclasses.

1. **Type**

The exact class that is used for class instantiation is called **type**.Normally, we define a class using a special syntax called the *class keyword*, but this syntax is a substitute for *type class*.

**class** **FoodType**(object):

**def** \_\_init\_\_(self, ftype):

self.ftype = ftype

**def** getFtype(self):

**return** self.ftype

Using **type** as a keyword instead of **class**

**def** init(self, ftype):

self.ftype = ftype

**def** getFtype(self):

**return** self.ftype

FoodType = type('FoodType', (object, ), {

'\_\_init\_\_': init,

'getFtype' : getFtype,

})

* The first argument is a string – *FoodType*. This string is assigned as the class name.
* The second argument is a tuple – *(object, )*. This tells that the *FoodType* class inherits from the object class. Here, the trailing comma helps the python interpreter to recognize it as a tuple.
* Here, the third argument is a dictionary that mentions the attribute of a class. In this case, the class has two methods – *init* and *getFtype.*

1. **Writing Metaclasses**

Metaclasses are classes that inherit directly from type.

**Creating Instance Objects in Python**

instance object is an individual object created from a class.

**Dynamic Attributes in Python**

Attributes that are **defined at runtime**, after creating the objects or instances.

**class** **GFG**:

**None**

**def** value():

**return** 10

g = GFG()

g.d1 = value //  reference to the value function

value.d1 = "Geeks"

print(value.d1)

print(g.d1() == value())

**Constructors in Python**

Special method that is called automatically when an object is created from a class.

\_\_new\_\_ : creates a new instance of the class

\_\_init\_\_ : sets up the instance's attributes after creation.

**Types of Constructors**

* 1. **Default Constructor**

A **default constructor** does not take any parameters other than **self**.

* 1. **Parameterized Constructor**

A**parameterized constructor** accepts arguments to initialize the object's attributes with specific values.

**Why Python Uses 'Self' as Default Argument**

The main reason Python uses self as the default argument is to make object-oriented programming explicit rather than implicit.

**class** **Car**:

**def** \_\_init\_\_(self, brand, model):

self.brand = brand

self.model = model

**def** display(self):

**return** self.brand, self.model

car1 = Car("Toyota", "Corolla")

print(car1.display())

* **self in \_\_init\_\_:**Used to assign values (brand and model) to the specific instance (car1).
* **self in display\_info:** Refers to the same car1 instance to access its attributes (brand and model).
* Python automatically passes car1 as the first argument to display.