1. In python everything is associated with type. In other programming languages, int, str, array are data type. But, in python, they are objects of the class int, str, list, dict. We can create user defined type by creating a class and then object of that class.

A Class is also an object, and just like any other object it’s a instance of something called Metaclass. A special class type creates these Class object. The type class is default metaclass which is responsible for making classes.

2. When defining a class and no metaclass is defined the default type metaclass will be used. If a metaclass is given and it is not an instance of type(), then it is used directly as the metaclass.

class A(type):

pass

class B(metaclass=A): # for class B, class A is a metaclass

pass

class C(B):

pass

print(type(A))

print(type(B))

print(type(C))

<class 'type'>

<class '\_\_main\_\_.A'>

<class '\_\_main\_\_.A'>

3. Both decorators and metaclass have some common things. When we want to aplly some common functionality to attributes of the class instances, both can be used. There are some problems which can be solved by decorators as well as by metaclasses. But there are a few problems that can only be solved by metaclasses. Use of metaclass affects its children while the decorator affects only the current class.

4. In this specific case—adding methods to a class when it's created—the choice between metaclasses and decorators is somewhat arbitrary. Decorators can be used to manage both instances and classes, and they intersect with metaclasses in the second of these roles.