1. What is a metaclass?

Ans.

In object-oriented programming, a metaclass is a class whose instances are classes. Just as an ordinary class defines the behavior of certain objects, a metaclass defines the behavior of certain classes and their instances. A class is itself an instance of a metaclass.

2. How do you declare the metaclass of a class?

Ans.

type is a default metaclass in python. But we can create a meta class in the following way:

In [54]:

class meta\_class(type):

pass

class ineuron(metaclass=meta\_class):

pass

In [55]:

type(ineuron)

Out[55]:

\_\_main\_\_.meta\_class

So here ineuron was a class but now it is an instance for meta\_class class

3. How do class decorators overlap with metaclasses for managing classes?

Ans.

Anything which we can do with a class decorator, can be done with a custom metaclass. We just need to apply the functionality of the "decorator function", i.e., the one that takes a class object and modifies it, in the course of the metaclass's new or init that make the class object.

Decorartors rebind a class name to a result which could be called and metaclass route the class creation through a callable object but both hooks can be used for similar class objects. Meta class augment a class after they create it. Decorator simply augment and return the original class object. Decorators may have a slight disadvantage in this role if a new class must be defined because original class has been already created

4. How do class decorators overlap with metaclasses for managing instances?

Ans.

We can use both class decorator and metaclasses to manage class instances by inserting a wrapper object to catch the instance creation calls. Decorators may rebind the class name to a cllabale run on instance creation that retains the original class object. Metaclasses can also do the same but they might have some disadvantage as they need to create a new object for that