Q1. What is the concept of a metaclass?

The classes that generate other classes are defined as metaclasses.

Metaclasses are classes that inherit directly from type. The method that custom metaclasses should implement is the \_\_new\_\_ method. The arguments mentioned in the \_\_new\_\_  method of metaclasses reflects in the \_\_new\_\_ method of type class. It has four positional arguments. They are as follows:

1. The first argument is the metaclass itself.
2. The second argument is the class name.
3. The third argument is the  superclasses (in the form of tuple)
4. The fourth argument is the attributes of class (in the form of dictionary)

Q2. What is the best way to declare a class's metaclass?

A way to declare a class’ metaclass is by using **metaclass** keyword in class definition.

Example:

**class** meta(type):

**pass**

**class** class\_meta(metaclass**=**meta):

**pass**

print(type(meta))

print(type(class\_meta))

Q3. How do class decorators overlap with metaclasses for handling classes?

Anything you can do with a class decorator, you can of course do with a custom metaclasses (just 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!).

Q4. How do class decorators overlap with metaclasses for handling instances?

 Anything you can do with a class decorator, you can of course do with a custom metaclass (just 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!).