Q1. What is the concept of a metaclass?

Sol. A metaclass is a class used to create a class. Normal classes are instances of the type class by default. Metaclasses are usually subclasses of the type class, which redefines class creation protocol methods in order to customize the class creation call issued at the end of a class statement; they typically redefine the methods \_\_new\_\_ and \_\_init\_\_ to tap into the class creation protocol. Metaclasses can also be coded other ways—as simple functions, for example—but they are responsible for making and returning an object for the new class.

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

Sol. In Python 3.0 and later, use a keyword argument in the class header line: class C(metaclass=*M*). In Python 2.X, use a class attribute instead: \_\_metaclass\_\_ = *M*. In 3.0, the class header line can also name normal superclasses (a.k.a. base classes) before the metaclass keyword argument.

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

Sol. Because both are automatically triggered at the end of a class statement, classdecorators and metaclasses can both be used to manage classes. Decorators rebind a class name to a callable’s result and metaclasses route class creation through a callable, but both hooks can be used for similar purposes. To manage classes, decorators simply augment and return the original class objects. Metaclasses augment a class after they create it.

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

Sol. Because both are automatically triggered at the end of a class statement, class decorators and metaclasses can both be used to manage class instances, by inserting a wrapper object to catch instance creation calls. Decorators may rebind the class name to a callable run on instance creation that retains the original class object. Metaclasses can do the same, but they must also create the class object, so their usage is somewhat more complex in this role