Q1. What is the concept of a metaclass?

Ans: A metaclass is a class used to create a class. Normal new-style 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 always responsible for making and returning an object for the new class. Meta classes may have methods and data to provide behavior for their classes too and constitute a secondary pathway for inheritance search but their attributes are accessible only to their class instances, not to their instance's instances.

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

Ans: In Python 3.X, use a keyword argument in the class header line: class C(meta class=M). In Python 2.X, use a class attribute instead: __metaclass__ = M. In 3.X, the class header line can also name normal superclasses before the metaclass keyword argument; in 2.X we generally should derive from object too, though this is sometimes optional.

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

Ans: Because both are automatically triggered at the end of a class statement, class decorators 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. Decorators may have a slight disadvantage in this role if a new class must be defined, because the original class has already been created.

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

Ans: Because both are automatically triggered at the end of a class statement, we can use both class decorators and metaclasses to manage class instances, by inserting a wrapper (proxy) 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 may have a slight disadvantage in this role, because they must also create the class object.