Q1. What is the concept of a metaclass?

Ans : A metaclass in Python is a class of a class that defines how a class behaves. A class is itself an instance of a metaclass. A class in Python defines how the instance of the class will behave.

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

Ans :

Metaclasses in Python 3

The syntax to set the metaclass has been changed in Python 3:

class Foo(object, metaclass=something):

...

i.e. the \_\_metaclass\_\_ attribute is no longer used, in favor of a keyword argument in the list of base classes.

The behavior of metaclasses however stays largely the same.

One thing added to metaclasses in Python 3 is that you can also pass attributes as keyword-arguments into a metaclass, like so:

class Foo(object, metaclass=something, kwarg1=value1, kwarg2=value2):

...

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

Ans :

Just like with metaclasses, because the decorator returns the original class, instances are made from it, not from a wrapper object. In fact, instance creation is not intercepted at all. ... Decorators can be used to manage both instances and classes, and they intersect with metaclasses in the second of these roles.

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

Ans : Just like with metaclasses, because the decorator returns the original class, instances are made from it, not from a wrapper object. In fact, instance creation is not intercepted at all. ... Decorators can be used to manage both instances and classes, and they intersect with metaclasses in the second of these roles.