Q1. What is the concept of 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. Not all object-oriented programming languages support metaclasses.

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

ANS:

Creating Custom Metaclasses

1. class MyMeta(type): pass class MyClass(metaclass=MyMeta): pass class MySubclass(MyClass): pass. ...
2. print(type(MyMeta)) print(type(MyClass)) print(type(MySubclass)) ...
3. <class 'type'> <class '\_\_main\_\_.MyMeta'> <class '\_\_main\_\_.MyMeta'>

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

ANS:

Decorators are much, much simpler and more limited -- and therefore should be preferred whenever the desired effect can be achieved with either a metaclass or a class decorator.

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!-).

There are many things you can do in a custom metaclass but not in a decorator (unless the decorator internally generates and applies a custom metaclass, of course -- but that's cheating;-)... and even then, in Python 3, there are things you can only do with a custom metaclass, not after the fact... but that's a pretty advanced sub-niche of your question, so let me give simpler examples).

For example, suppose you want to make a class object X such that print X (or in Python 3 print(X) of course;-) displays peekaboo!. You cannot possibly do that without a custom metaclass, because the metaclass's override of \_\_str\_\_ is the crucial actor here, i.e., you need a def \_\_str\_\_(cls): return "peekaboo!" in the custom metaclass of class X.

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

ANS:

Metaclasses operate at the lower level and allow you to change the structure or behavior of the class, like the class methods, attributes, and inheritance. Decorators, however, are used to modify the functions' behavior. They allowed you to add functionality to the existing functions without changing the code.