Q1. What is the concept of a metaclass?

Ans: A metaclass in [Python](https://www.datacamp.com/tutorial/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. In order to understand metaclasses well, one needs to have prior experience working with Python classes.

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

class MyMeta(type):

pass

class MyClass(metaclass=MyMeta):

pass

class MySubclass(MyClass):

pass

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 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.

The same applies to all magic methods, i.e., to all kinds of operations as applied to the class object itself (as opposed to, ones applied to its *instances*, which use magic methods as defined in the class -- operations on the class object itself use magic methods as defined in the metaclass).

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

The method decorator marks the method as one that is of interest by adding a "use\_class" attribute - functions and methods are also objects, so you can attach additional metadata to them.

After the class has been created the class decorator then goes through all the methods and does whatever is needed on the methods that have been marked.

If you want all the methods to be affected then you could leave out the method decorator and just use the class decorator.