Q1. What is the concept of a metaclass?

A metaclass is the class of a class. A class defines how an instance of the class (i.e. an object) behaves while a metaclass defines how a class behaves. A class is an instance of a metaclass.

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

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

class Foo(object, metaclass=something):

...

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?

[class decorators](https://www.pythonstudio.us/object-oriented/class-decorators.html)  sometimes overlap with metaclasses in terms of functionality. Although they are typically used for managing or augmenting instances, class decorators can also augment classes, independent of any created instances.

In this specific case—adding methods to a class when it's created—the choice between metaclasses and decorators is somewhat arbitrary. Decorators can be used to manage both instances and classes, and they intersect with metaclasses in the second of these roles.

However, this really addresses only one operational mode of metaclasses. As we'll see, decorators correspond to metaclass\_\_init\_methods in this role, but metaclasses have additional customization hooks. As we'll also see, in addition to class initialization, metaclasses can perform arbitrary construction tasks that might be more difficult with decorators.

Moreover, although decorators can manage both instances and classes, the converse is not as direct—metaclasses are designed to manage classes, and applying them to managing instances is less straightforward

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

\*\*\*Repeated\*\*\*