Q1. What is the concept of a metaclass?
Answer: Metaclass is a class of a class which describes how the class behaves. A class in itself is an instance of metaclass.

The __class__ attribute the type of the an instance :-
hlp="help"

hlp.__class__

>> str

When we check the type of "str" we get type

str.__class__
 >>> type

This shows type is a metaclass that defines other classes. If we do __class__.__class__ we get the metaclass of a variable

hlp.__class__.__class__
>> type

Q2. What is the best way to declare a class's metaclass?
Answer: Using the __new__ and __init__ method.
The __new__ method is a way to define dictionary tuples before the class is created. Return value of __new__ is an instance of cls class.

```
class MetaOne(type):
    def __new__(cls, name, bases, dict):
        pass

class MetaTwo(type):
    def __init__(self, name, bases, dict):
        pass
```

Q3. How do class decorators overlap with metaclasses for handling classes?
Answer: Anything you can do with a class decorator, you can also do with a custom metaclass.
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

Q4. How do class decorators overlap with metaclasses for handling instances?
Answer: Because both metaclass and decorators are triggered at the end of a class statement, we can use both decorators and metclass to manage class instances by inserting a wrapper to catch instance creation calls.
Decorators may bind the class name to a callable on instance creation that retains the original class, Metaclasses can do the same but they must also create the class object