Q1. What is the concept of a metaclass?

The term metaprogramming refers to the potential for a program to have knowledge of or manipulate itself. Python supports a form of metaprogramming for classes called metaclasses.

Metaclasses are an esoteric OOP concept, lurking behind virtually all Python code. You are using them whether you are aware of it or not. For the most part, you don’t need to be aware of it. Most Python programmers rarely, if ever, have to think about metaclasses.

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

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. In order to understand metaclasses well, one needs to have prior experience working with Python classes.

class TestClass():

pass

my\_test\_class = TestClass()

print(my\_test\_class)

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

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.

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

Whenever a \_\_metaclass\_\_ is used, one could also use a decorator to get effectively the same result. This section discusses this topic.

For an example we use the concept of a Bunch, as discussed in Alex Martelli’s excellent book Python in a Nutshell.