Q1. The key difference between `__getattr__` and `__getattribute__` is that `__getattr__` is called only when the attribute is not found in the normal way, whereas `__getattribute__` is called for every attribute access. `__getattribute__` is called even if the attribute is found in the instance or the class. It is possible to get into infinite recursion if you try to access the same attribute in `__getattribute__`. To avoid this, you can use `super().__getattribute__()` to call the parent class implementation.

Q2. Properties and descriptors are both used to manage attribute access in Python, but they work in slightly different ways. A property is a special type of attribute that is computed dynamically, whenever it is accessed. A descriptor, on the other hand, is a class that defines methods for getting, setting, and deleting an attribute. Descriptors can be used to implement properties, but they can also be used for other purposes, such as type checking and validation.

Q3. The main difference in functionality between `__getattr__` and `__getattribute__` is that `__getattr__` is only called when an attribute is not found in the normal way, whereas `__getattribute__` is called for every attribute access. This means that `__getattribute__` can be used to intercept and modify attribute access in a more fine-grained way.

Similarly, the main difference between properties and descriptors is that properties are a simpler mechanism for implementing computed attributes that can be accessed like regular attributes, whereas descriptors are a more general mechanism for managing attribute access. Descriptors can be used to implement properties, but they can also be used for other purposes, such as type checking and validation.