Calling a Descriptor

The most common method of calling a descriptor is for the descriptor to be invoked automatically when you access an attribute. A typical example would be `my_obj.attribute_name`. This will cause your object to look up `attribute_name` in the `my_obj` object. If your `attribute_name` happens to define __get__(), then `attribute_name.__get__(my_obj)` will get called. This all depends on whether your instance is an object or a class.

The magic behind this lies in the magic method known as __getattribute__, which will turn `my_obj.a` into this: `type(my_obj).__dict__['a'].__get__(a, type(a))`. You can read all about the implementation in Python's documentation here: https://docs.python.org/3/howto/descriptor.html.

According to said documentation, there are a few points to keep in mind in regards to calling a descriptor:

> - The descriptor is invoked via the default implementation of the __getattribute__ method
> - If you override __getattribute__, this will prevent the descriptor from getting automatically called
> - object.__getattribute__() and type.__getattribute__() don't call __get__() the same way
> - A data descriptor will always, ALWAYS override instance dictionaries
> - The non-data descriptor can be overridden by instance dictionaries.

More information on how all this works can be found in Python's data model (https://docs.python.org/3/reference/datamodel.html#object.__getattribute__), the Python source code and in Guido van Rossum's document, "Unifying types and class in Python", which can be found here:

https://www.python.org/download/releases/2.2.3/descrintro/#cooperation