**FSDS MAY BATCH 2022(Python Assignment -11)**

**Submitted by: Shubham Tiwari**

Q1. What is the concept of a metaclass?

Ans: 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. We basically write in a following manner:

class TestClass():

pass

my\_test\_class = TestClass()

print(my\_test\_class)

Q2: What is the best way to declare a class’s metaclass?

Ans:The best way to declare a class metaclass is :

class TestClass():

pass

my\_test\_class = TestClass()

print(my\_test\_class)

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

Ans: Decorators are much, much simpler and more limited -- and therefore should be preferred whenever the desired effect can be achieved with either a metaclass or a class decorator.

Anything we can do with a class decorator, you can of course do with a custom metaclass (just apply the functionality of the "decorator function", i.e., the one that takes a class object and modifies it, in the course of the metaclass's \_\_new\_\_ or \_\_init\_\_ that make the class object).

There are many things we can do in a custom metaclass but not in a decorator (unless the decorator internally generates and applies a custom metaclass, of course -- but that's cheating;-) and even then, in Python 3.X, there are things we can only do with a custom metaclass.

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

Ans: Just like with metaclasses, because the decorator returns the original class, instances are made from it, not from a [wrapper object](https://www.pythonstudio.us/object-oriented/oop-and-delegation-wrapper-objects.html). In fact, instance creation is not intercepted at all. For example, suppose we want to make a class object X such that print X (or in Python 3 print(X) of course;-) displays peekaboo!. We cannot possibly do that without a custom metaclass, because the metaclass's override of \_\_str\_\_ is the crucial actor here, i.e., we need a def \_\_str\_\_(cls): return "peekaboo!" in the custom metaclass of class X.

The same applies to all magic methods, i.e., to all kinds of operations as applied to the class object itself (as opposed to, ones applied to its instances, which use magic methods as defined in the class -- operations on the class object itself use magic methods as defined in the metaclass).