Python Advanced Assignment 11

Q1. What is the concept of a metaclass?

Ans: Metaclass in Python is a class of a class that defines how a class behaves. A class is itself a instance of Metaclass, and any Instance of Class in Python is an Instance of type metaclass. E.g. Type of of int, str, float, list, tuple and many more is of metaclass type.

```
# our metaclass
class MultiBases(type):
    # overriding __new__ method
    def __new__(cls, clsname, bases, clsdict):
        # if no of base classes is greater than 1
        # raise error
        if len(bases)>1:
            raise TypeError("Inherited multiple base classes!!!")
            # else execute __new__ method of super class, ie.
            # call __init__ of type class
            return super().__new__(cls, clsname, bases, clsdict)
        # metaclass can be specified by 'metaclass' keyword argument
        # now MultiBase class is used for creating classes
        # this will be propagated to all subclasses of Base
        class Base(metaclass=MultiBases):
            pass
        # no error is raised
        class A(Base):
            pass
        # no error is raised
        class B(Base):
            pass

        # This will raise an error!
        class C(A, B):
            pass
```

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

Ans: A way to declare a class' metaclass is by using **metaclass** keyword in class definition.

```
class meta(type):
    pass
class class_meta(metaclass=meta):
    pass
print(type(meta))
print(type(class_meta))
```

<class 'type'>
<class '__main__.meta'>

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

Ans: Anything you can do with a class decorator, you can of course do with a custom metaclasses (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!).

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

Ans: Anything you 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!).