### **Lab 5: Metaclasses in Python**

**Lab Exercise: Modifying Attributes with a Metaclass**

In this exercise, we'll create a metaclass that automatically converts all string attributes to uppercase in a class definition.

Step 1: Define the Metaclass

class UppercaseAttributesMeta(type):

def \_\_new\_\_(cls, name, bases, attrs):

uppercase\_attrs = {}

for attr\_name, attr\_value in attrs.items():

if isinstance(attr\_value, str):

uppercase\_attrs[attr\_name] = attr\_value.upper()

else:

uppercase\_attrs[attr\_name] = attr\_value

return super().\_\_new\_\_(cls, name, bases, uppercase\_attrs)

Step 2: Define a Class Using the Metaclass

Now, let's define a class that uses our custom metaclass:

class UppercaseClass(metaclass=UppercaseAttributesMeta):

name = "hello"

description = "this is a description"

def display(self):

print(self.name, self.description)

Step 3: Create Instances and Test

obj = UppercaseClass()

print(obj.name) # Output: HELLO

print(obj.description) # Output: THIS IS A DESCRIPTION

obj.display() # Output: HELLO THIS IS A DESCRIPTION

Explanation:

In this exercise, we created a metaclass called UppercaseAttributesMeta. This metaclass modifies the attributes of any class that uses it by converting string attribute values to uppercase. We applied this metaclass to the UppercaseClass, and when we create an instance of UppercaseClass, the string attributes (name and description) are automatically converted to uppercase.

This example provides a simplified introduction to metaclasses and showcases how they can modify class attributes during class creation.

**Happy coding!**