### **Lab 3: Python Metaclass and Abstract Base Class**

**Exercise: Python Metaclass and Abstract Base Class**

Creating a lab exercise that combines abstract classes and metaclasses in Python can provide students with a deep understanding of both concepts and how they can be used to control class creation and behavior. In this exercise, Participants will create a simple framework for defining and managing plugins using abstract classes and metaclasses.

**Exercise Description:**

Imagine you are developing a plugin system for an application. You want to ensure that all plugins adhere to a common interface while allowing plugin developers to create custom plugins. To achieve this, we will create an abstract base class to define the required plugin methods and a metaclass to manage the registration of plugins.

**Instructions:**

1. **Create an Abstract Base Class (ABC):** Start by defining an abstract base class called Plugin with the following abstract methods:

* name(self): An abstract method that returns the name of the plugin.
* execute(self, input\_data): An abstract method that represents the main functionality of the plugin. It takes an input\_data parameter as input.

1. **Create a Metaclass for Plugin Registration:**

* Create a metaclass called PluginMeta that will be responsible for managing the registration of plugin classes.
* Implement the \_\_init\_\_ method of the metaclass to create an empty list plugins to store registered plugin classes.

1. **Implement the register Method:**

* In the PluginMeta metaclass, implement a class method called register that takes a plugin class as an argument.
* Inside the register method, append the plugin class to the plugins list.

1. **Create Concrete Plugin Classes:**

* Create at least two concrete plugin classes that inherit from the Plugin abstract base class. Each class should implement the name and execute methods according to the specific functionality of the plugin.

**Register Plugin Classes:**

* Use the @PluginMeta.register decorator to register the concrete plugin classes with the metaclass.

**Test the Framework:**

* Create instances of the registered plugin classes and demonstrate that they can be used interchangeably.
* Use a loop to execute all registered plugins with sample input data.

This exercise allows participants to practice creating abstract base classes, metaclasses, class registration, and polymorphism in Python. They can extend the framework by adding more concrete plugin classes to see how the registration system works and how different plugins can be executed uniformly.