1 Animals
Create two classes: one base class and two classes that inherit from it. Name the base class Mammal, and name the two other classes after any mammals you choose. The base class should have a field for a defined object called info and a method called fun fact, where info is a public field and fun fact is a public method that displays the info field. All classes should have a variable called type. In the constructor of the Mammal class, information about the object's type should be displayed. The inherited constructor from the base class should pass the info parameter from the inherited class.

2 Player and Monsters
Create an abstract class named GameObject. This class should have a constructor that takes the object's health points as its only argument. This class should also have a method that checks whether the object is "alive" (returns True if the health points are greater than zero) and an abstract method called .interact() that takes one argument (which will be explained later).

Three other classes must inherit from this class: Player, Monster, and Door, which represent the player, monster, and door respectively. None of the created classes overrides the constructor, the constructor from the base class will be used.

In the Player class, the .interact() method must be overridden but does not necessarily need to have an implementation (if we use the @abstractmethod decorator and do not override this method in the inherited class, we will not be able to create an object of this class). We assume that the argument passed to this method will always be an object of the Player class.

In the Door class, the .interact() method must display information about the player passing through the door.

In the Monster class, the .interact() method must reduce the health points of the object passed as an argument (i.e., the player) by 10, then set the health points of this monster to 0, and display information about the monster being killed by the player.

After creating the classes described above, we need to write code that simulates a simple game: we create a player object (an object of the Player class), then create a list and randomly fill it with a specified number of objects from the Monster or Door classes (using a condition that allows us to randomly select an object of which class to add to the list).

Next, we write a loop that goes through the objects in this list and calls the .interact() method on each of them, passing the previously created player object as an argument (obj.interact(player)).

At the end of each iteration, we check whether the player is still alive (the method for this was implemented in the base class). In case the player died, we display information about it and terminate the loop.

3 Equations
Create an abstract class representing an equation with given coefficients. The abstract equation class must have a constructor that takes a list of equation coefficients and a method .solve() that must solve the equation (no implementation will be provided in the abstract class).

Then create two classes that inherit from the equation class. These classes must represent a linear and quadratic equation. The .solve() method must be overridden and implemented to solve the equation with the coefficients provided in the constructor arguments. The constructor of the base class must also be overridden to check if the number of coefficients provided matches the equation (check if the list has two coefficients for a linear equation or three coefficients for a quadratic equation).