Composition

In this lesson, you'll learn how to achieve composition in Python.

We'll cover the following

Composition is the practice of accessing other class objects in your class. In such a scenario, the class which creates the object of the other class is known as the *owner* and is responsible for the lifetime of that object.

Composition relationships are **Part-of** relationships where the *part* must constitute a segment of the whole object. We can achieve composition by adding smaller parts of other classes to make a complex unit.

So, what makes composition so unique?

> In composition, the lifetime of the owned object depends on the lifetime of the owner.

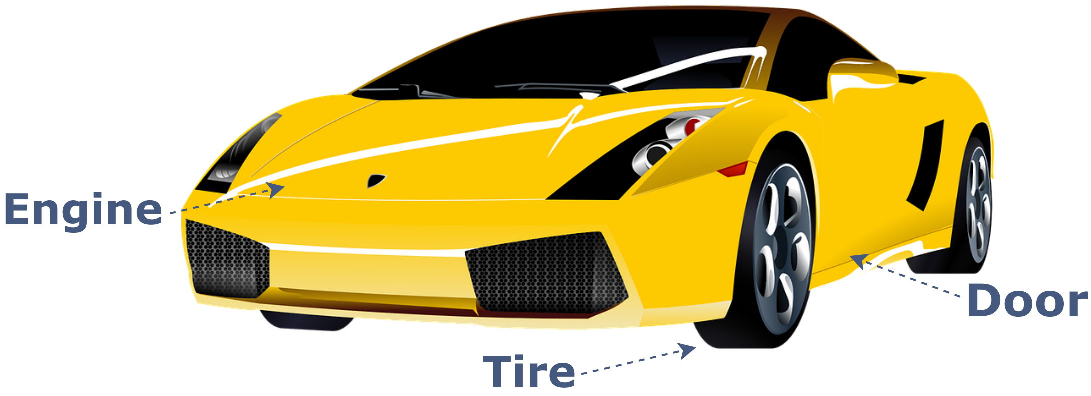
Example

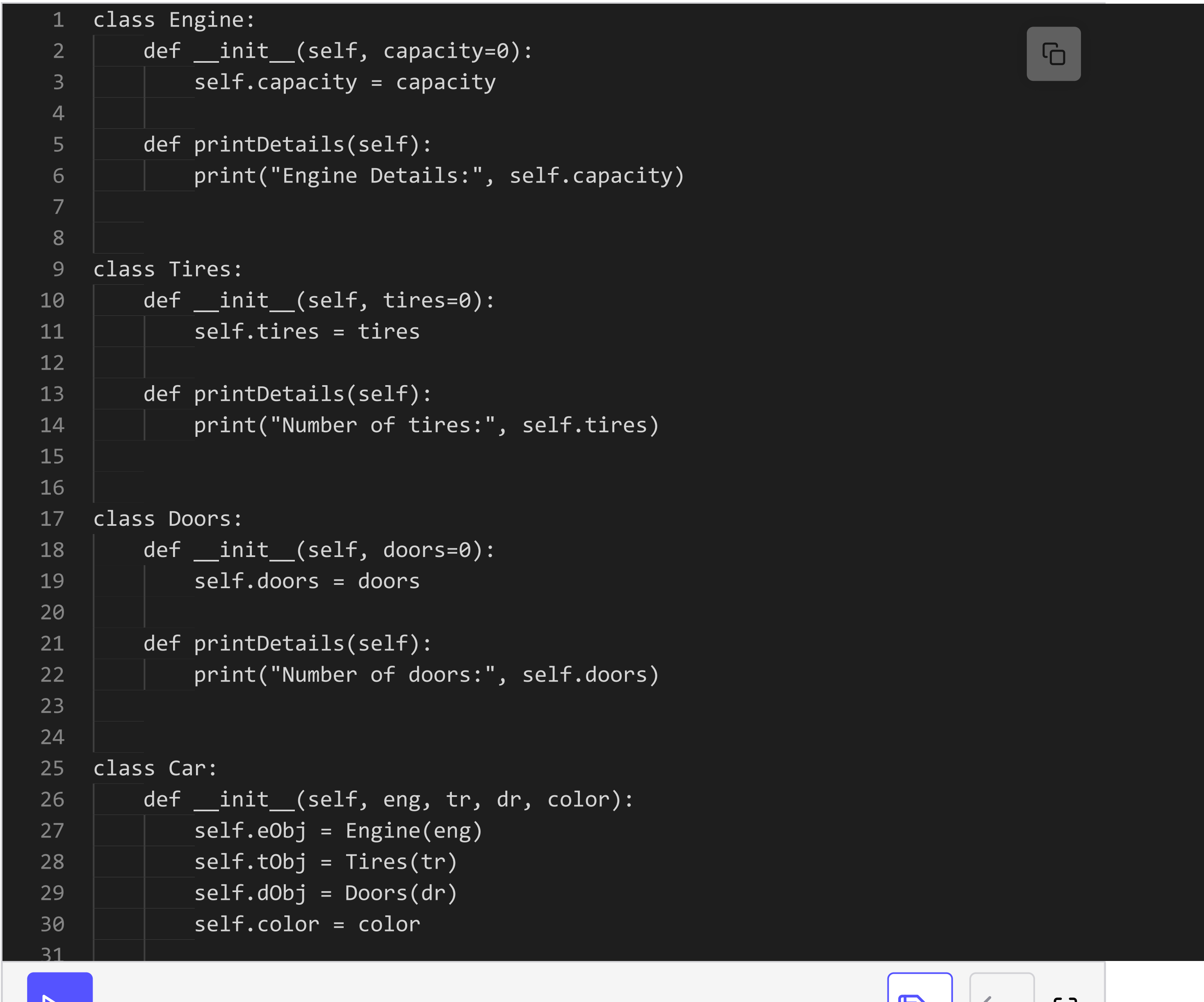
A `car` is composed of an *engine*, *tires*, and *doors*. In this case, a `Car` owned these objects, so a `Car` is an *Owner* class and `tires`, `doors`, and `engine` classes are *Owned* classes.

Implementation

Let’s look at the implementation of `Car` class for better understanding:

A car is composed of engine, tires and doors.





```
class Engine:
    def __init__(self, capacity=0):
        self.capacity = capacity

    def printDetails(self):
        print("Engine Details:", self.capacity)


class Tires:
    def __init__(self, tires=0):
        self.tires = tires

    def printDetails(self):
        print("Number of tires:", self.tires)


class Doors:
    def __init__(self, doors=0):
        self.doors = doors

    def printDetails(self):
        print("Number of doors:", self.doors)


class Car:
    def __init__(self, eng, tr, dr, color):
        self.eObj = Engine(eng)
        self.tObj = Tires(tr)
        self.dObj = Doors(dr)
        self.color = color
```

We have created a `Car` class which contains the objects of `Engine`, `Tires`, and `Doors` classes. `Car` class is responsible for their lifetime, i.e., when Car dies, so does *tire*, *engine*, and *doors* too.

Now, let’s test your knowledge with a quick quiz!