Previous: Python Constructor

Next: Structure Your Project ›

Home » Classes and Objects in Python » Python Inheritance

Python Inheritance

February 12, 2022

In programming, it's considered good style to reuse as much code as possible. There's even a nice acronym for this practice, called DRY: Don't Repeat Yourself. Classes help you to avoid repeating code because you can write a class once and create many objects based on it. However, they also help you in another way when using Python inheritance.

Table of Contents [hide]

Beginners Python Course (2023)

Modules, Packages, And Virtual Environments (2023)

Inheritance in Python

We've already seen inheritance at work, but you may not have realized it yet. Remember how I told you about Python constructors and that every class has a constructor (`__init__`), even when you don't define one? It's because every class inherits from the most basic class in Python, called `object`:

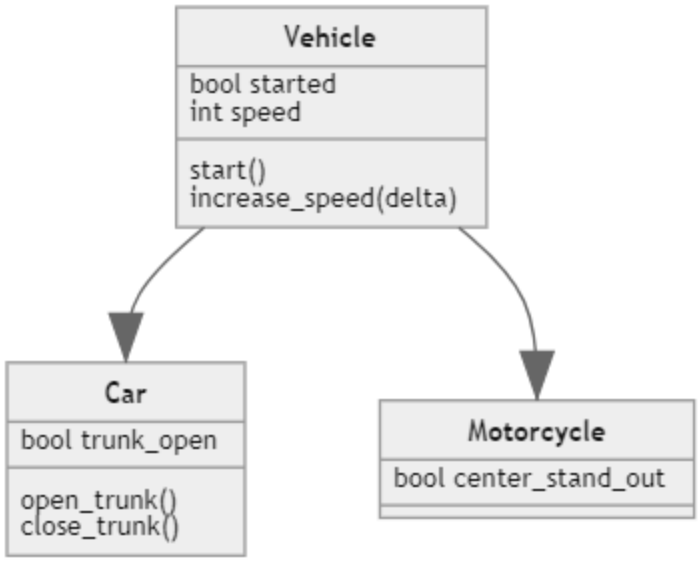
```
>>> dir(object)
['__class__', '__delattr__', '__dir__',
'__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__',
'__hash__', '__init__', '__init_subclass__',
'__le__', '__lt__', '__ne__', '__new__',
```

```
'__reduce__', '__reduce_ex__', '__repr__',
'__setattr__', '__sizeof__', '__str__',
'__subclasshook__']
```

When I told you 'everything in Python is an object', I really meant everything. That includes classes and as you can see we can use `dir()` on a class too; the `object` class. It reveals that `object` has an `__init__` method. Cool, isn't it?

Python inheritance example

Classes can inherit properties and functions from other classes, so you don't have to repeat yourself. Say, for example, we want our Car class to inherit some more generic functions and variables from a Vehicle class. While we're at it, let's also define a Motorcycle class. Schematically, it looks like this:



Python class inheritance

Inheritance maps to many real-life situations. Let's see inheritance in action, based on the class diagram above. We'll start with a generic `Vehicle` class:

```
class Vehicle:
```

```
    def __init__(self, started = False, speed = 0):
        self.started = started
        self.speed = speed
    def start(self):
        self.started = True
        print("Started, let's ride!")
    def stop(self):
        self.speed = 0
    def increase_speed(self, delta):
        if self.started:
            self.speed = self.speed + delta
            print("Vroooooom!")
        else:
            print("You need to start me first")
```

Now we can redefine our Car class, using inheritance:

```
class Car(Vehicle):
    trunk_open = False
    def open_trunk(self):
        self.trunk_open = True
    def close_trunk(self):
        self.trunk_open = False
```

Our car inherits all methods and variables from the Vehicle class but adds an extra variable and two methods to operate the trunk.

Overriding Python methods

Sometimes you want to override the inherited `__init__` function. To demonstrate, we can create a Motorcycle class. Most motorcycles have a center stand. We'll add the ability to either put it out or in on initialization:

```
class Motorcycle(Vehicle):
    def __init__(self, center_stand_out = False):
        self.center_stand_out = center_stand_out
        super().__init__()
```

When you override the constructor, the constructor from the parent class that we inherited is not called at all. If you still want that functionality, you have to call it yourself. This is done with `super()`: it returns a reference to the parent class, so we can call the parent class's constructor.

In this case, we added functionality for the center stand but removed the option to set the speed and started state in the constructor. If you want, you can add options for speed and started state too and pass those on to the `Vehicle` constructor.

Overriding other methods

Just like `__init__`, we can override other methods as well. For example, if you want to implement a motorcycle that doesn't start, you can override the start method:

```
class Motorcycle(Vehicle):
    def __init__(self, center_stand_out = False):
        self.center_stand_out = center_stand_out
        super().__init__()
    def start(self):
        print("Sorry, out of fuel!")
```

Keep learning

Here are some resources to dive deeper into this subject:

- The official Python guide on inheritance.

Get certified with our courses

Learn Python properly through small, easy-to-digest lessons, progress tracking, quizzes to test your knowledge, and practice sessions. Each course will earn you a downloadable course certificate.

Beginners Python Course (2023)

Modules, Packages, And Virtual Environments (2023)

NumPy Course: The Hands-on Introduction To NumPy (2023)

‹ Previous: Python Constructor

Next: Structure Your Project