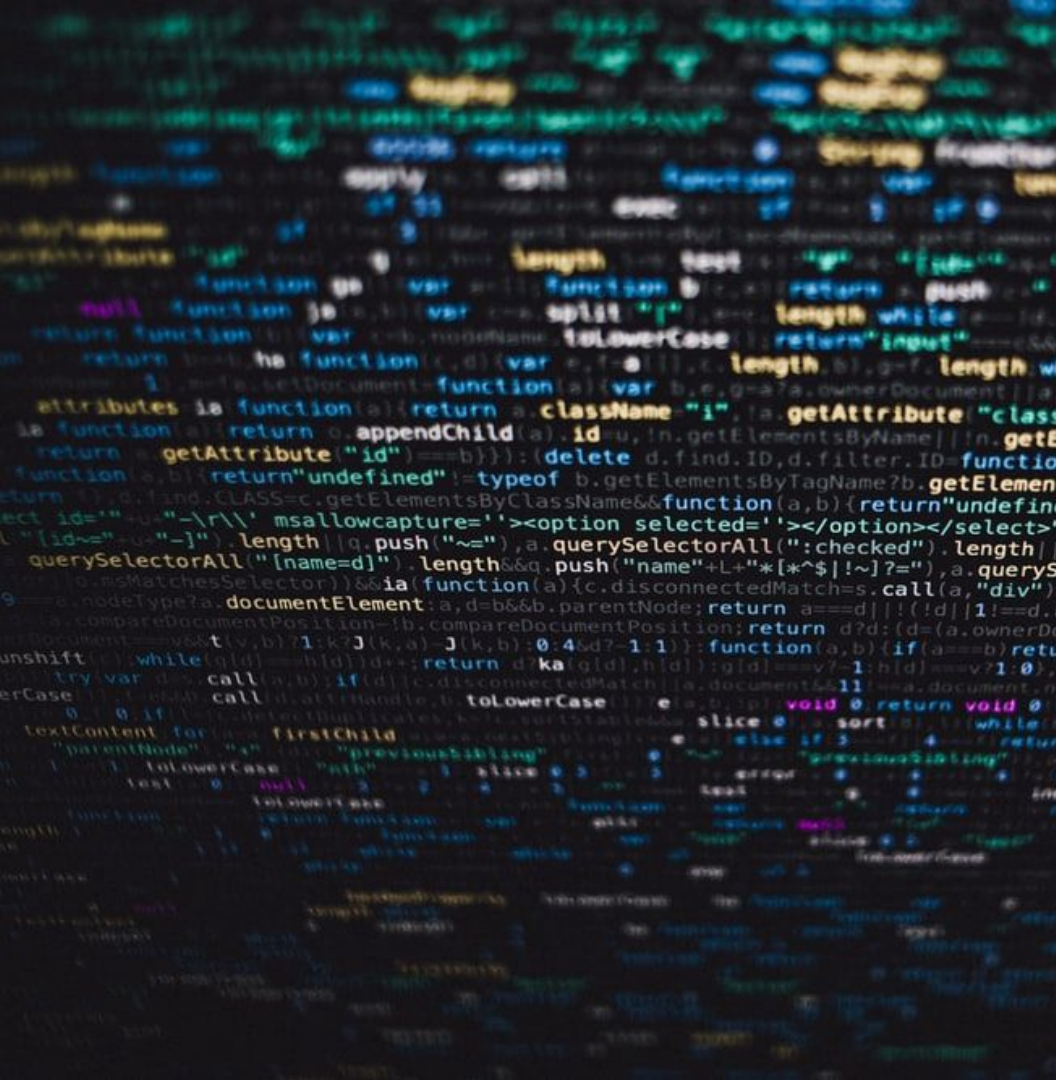
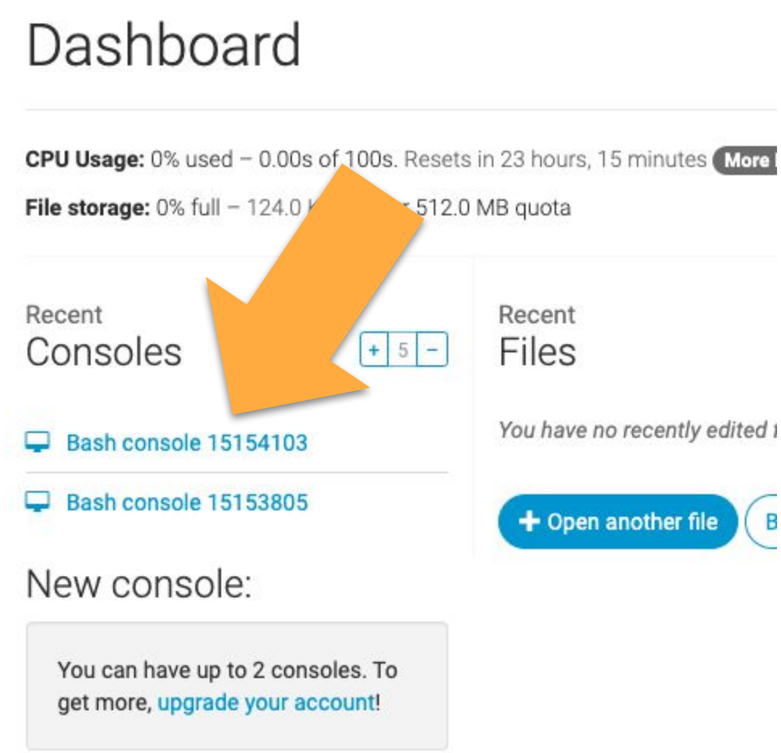


CODE CAMP

Day 007: More Classes and Objects



Dashboard

CPU Usage: 0% used – 0.00s of 100s. Resets in 23 hours, 15 minutes

File storage: 0% full – 124.0 k 512.0 MB quota

Recent
Consoles

Bash console 15154103

Bash console 15153805

New console:

You can have up to 2 consoles. To get more, upgrade your account!

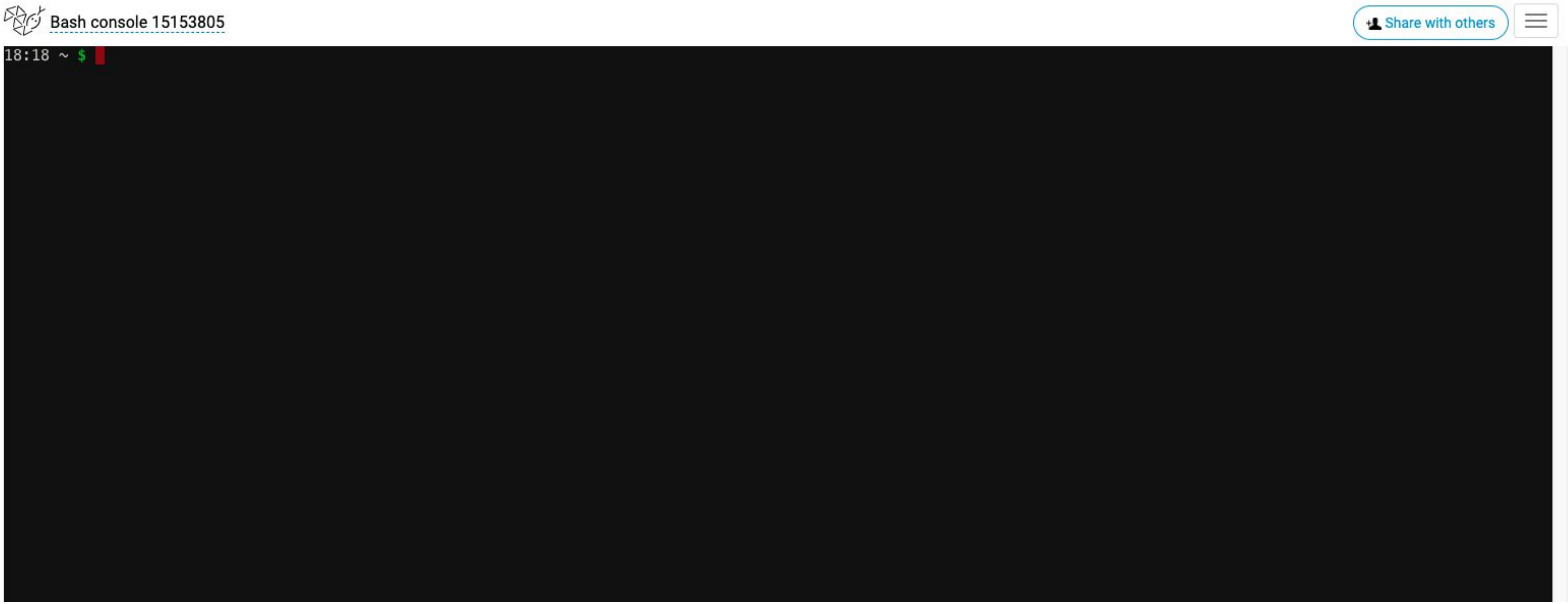
Recent
Files

You have no recently edited

.001 Log In to Python Anywhere

Visit PythonAnywhere.com and log in to your account. You can click on one of the bash consoles you logged in to previously. They are listed under “Recent Consoles.”

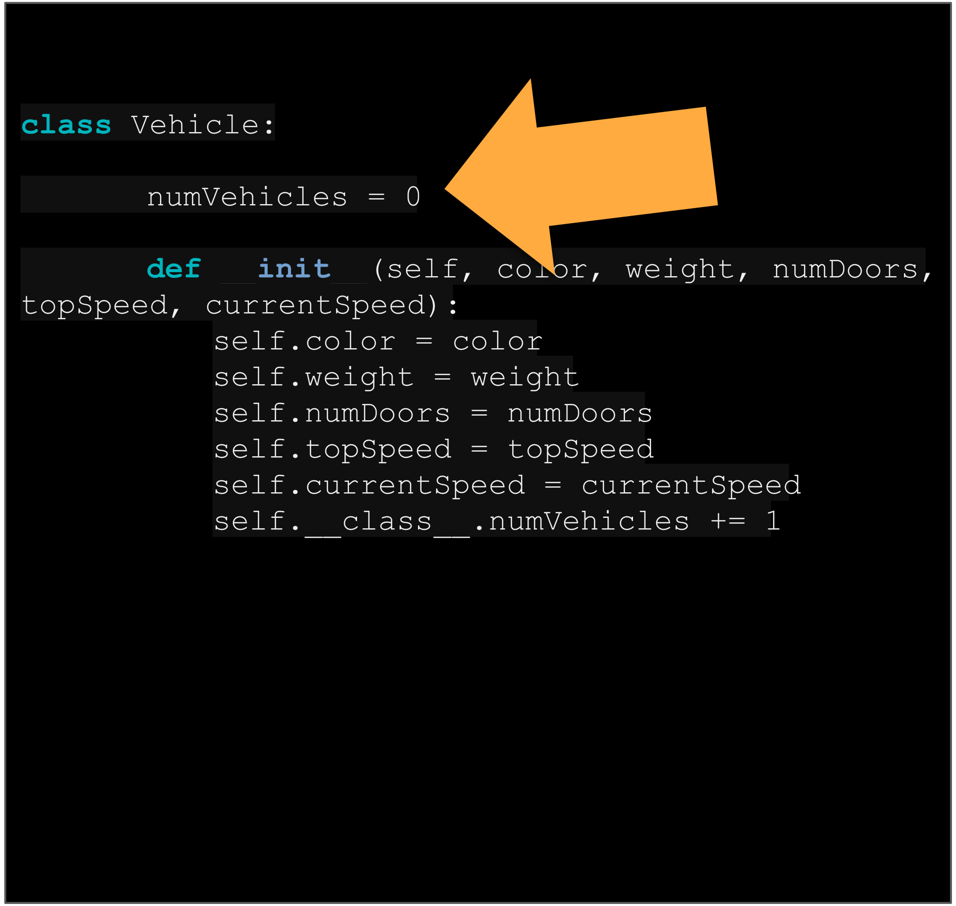
If your command line is cluttered from your previous work you can type the **clear** command to clear the console visual history.



```
18:18 ~ $
```

.001 Log In to Python Anywhere

If you’ve done everything correctly, your screen should appear similar to the image above. At your command prompt, type the command **nano** to open your text editor.



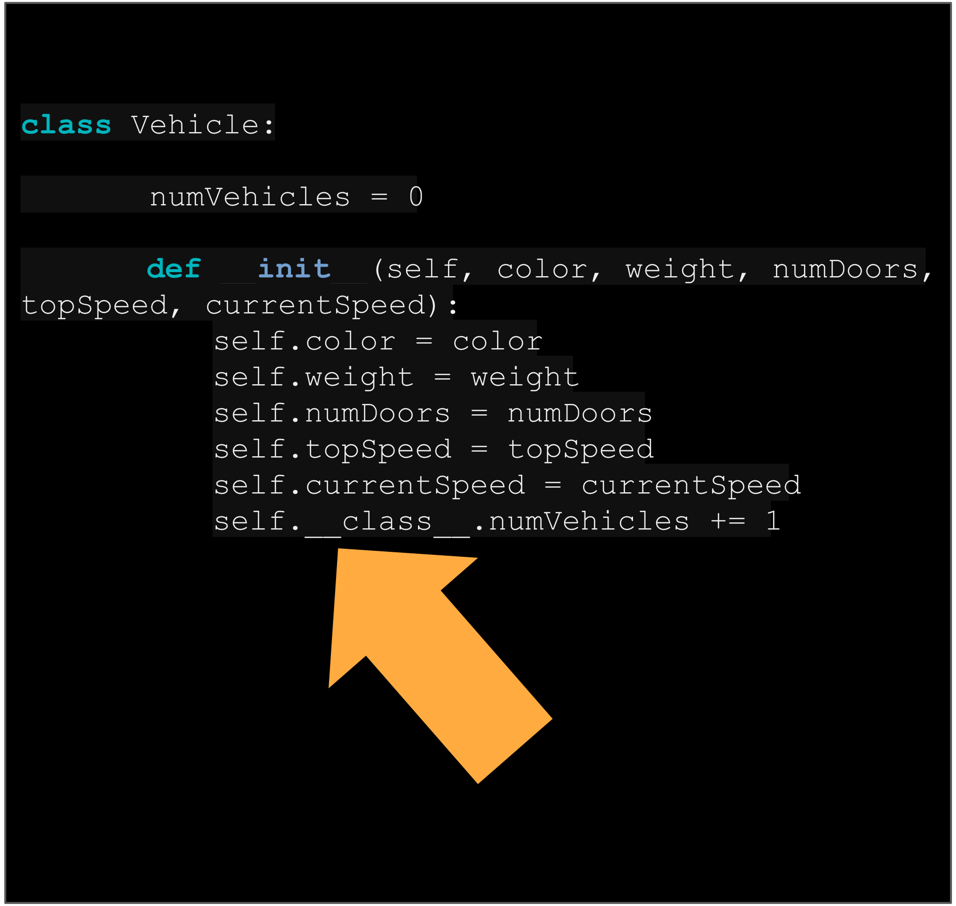
```
class Vehicle:

        numVehicles = 0

        def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
            self.color = color
            self.weight = weight
            self.numDoors = numDoors
            self.topSpeed = topSpeed
            self.currentSpeed = currentSpeed
            self.__class__.numVehicles += 1
```

.002 Class Attributes

Inside classes we have previously created properties that were instance members of the class. This means that when an instance of the class is created, these properties become part of the instance object.

Notice in the definition of the **Vehicle** class on the left that the property **numVehicles** has been added. It is not, however, part of the initializer, and does not have a value assigned to it at initialization.

Instead, it is incremented when initialization occurs. **numVehicles** is known as a class attribute or, sometimes, a static attribute.

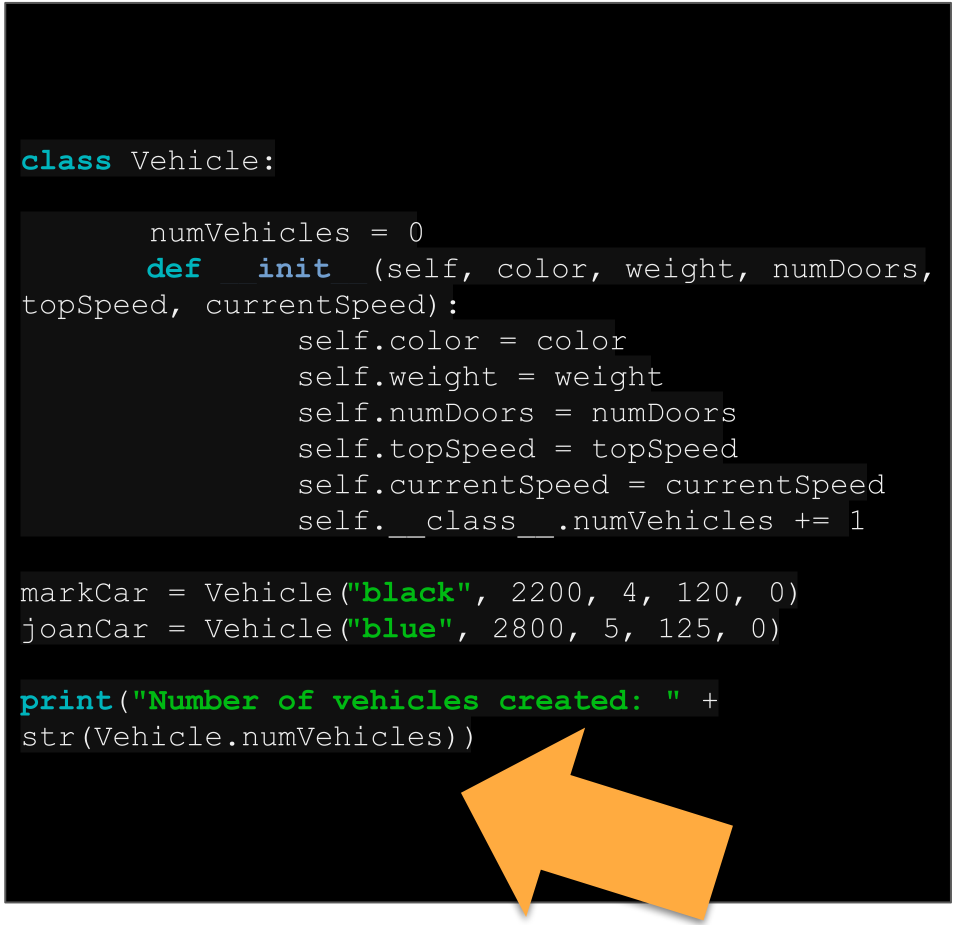


```
class Vehicle:

       numVehicles = 0

       def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
          self.color = color
          self.weight = weight
          self.numDoors = numDoors
          self.topSpeed = topSpeed
          self.currentSpeed = currentSpeed
          self.__class__.numVehicles += 1
```

.002 Class Attributes

Class attributes are useful for doing internal accounting with your class. You might, at some point during your program’s execution, might need to know the number of instances of a class created. The **numVehicles** class attribute will tell you just that.

Note that the class attribute is accessed through **self.__class__.numVehicles** from inside the class itself.

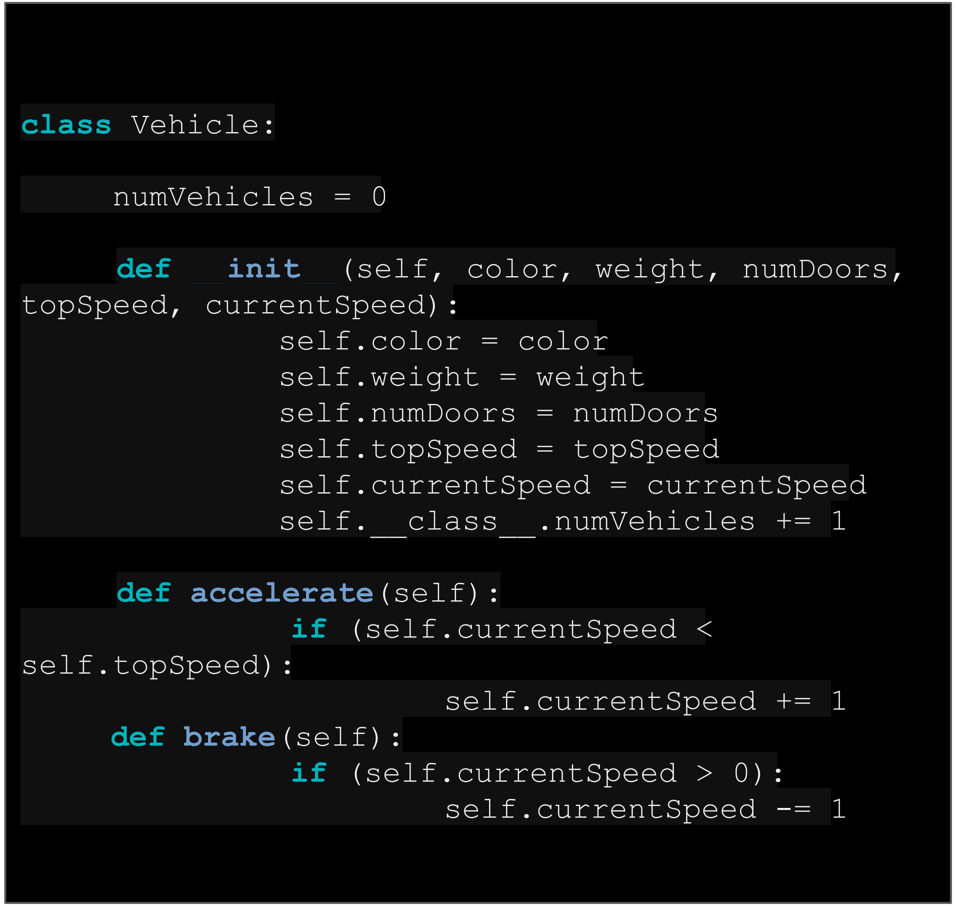


```
class Vehicle:

       numVehicles = 0
       def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
               self.color = color
               self.weight = weight
               self.numDoors = numDoors
               self.topSpeed = topSpeed
               self.currentSpeed = currentSpeed
               self.__class__.numVehicles += 1

markCar = Vehicle("black", 2200, 4, 120, 0)
joanCar = Vehicle("blue", 2800, 5, 125, 0)

print("Number of vehicles created: " +
str(Vehicle.numVehicles))
```

.002 Class Attributes
We can access the class attributes outside the class as well, using the familiar dot notation.

However, we don’t access the class attribute through the instances. The class attribute is access through the class name, **Vehicle**.

Key in the example and see if you get the result you expect when you run it on the command line. What happens if you try to access the **numVehicles** class attribute through an instance name like, **joanCar.numVehicles?** Discuss in the code camp Slack forum.

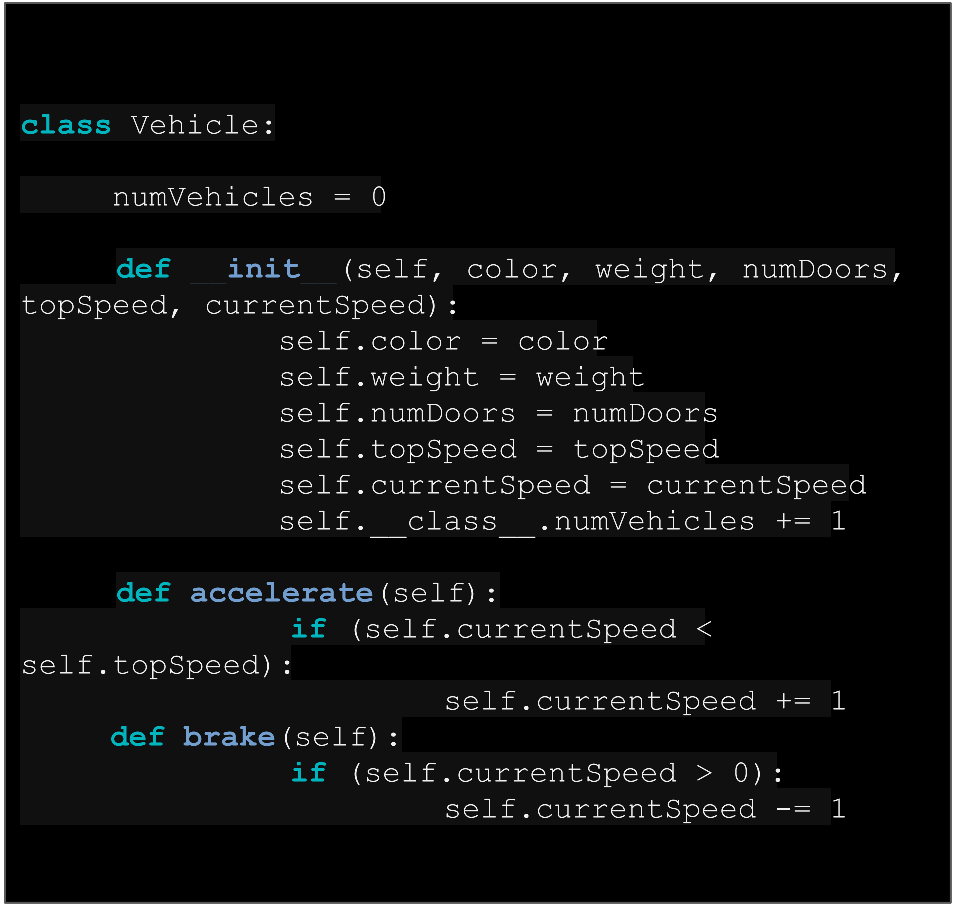


```
class Vehicle:

     numVehicles = 0

     def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
              self.color = color
              self.weight = weight
              self.numDoors = numDoors
              self.topSpeed = topSpeed
              self.currentSpeed = currentSpeed
              self.__class__.numVehicles += 1

     def accelerate(self):
               if (self.currentSpeed <
self.topSpeed):
                       self.currentSpeed += 1
     def brake(self):
               if (self.currentSpeed > 0):
                       self.currentSpeed -= 1
```

.003 Parent and Child Classes

Let’s update the code for our **Vehicle** class according the the code on the left. As always, ensure that your indentation is even so that code blocks can be properly identified by the Python interpreter.

Even though this code won’t output anything yet, run it with the **python** command to make sure there are no syntax errors.

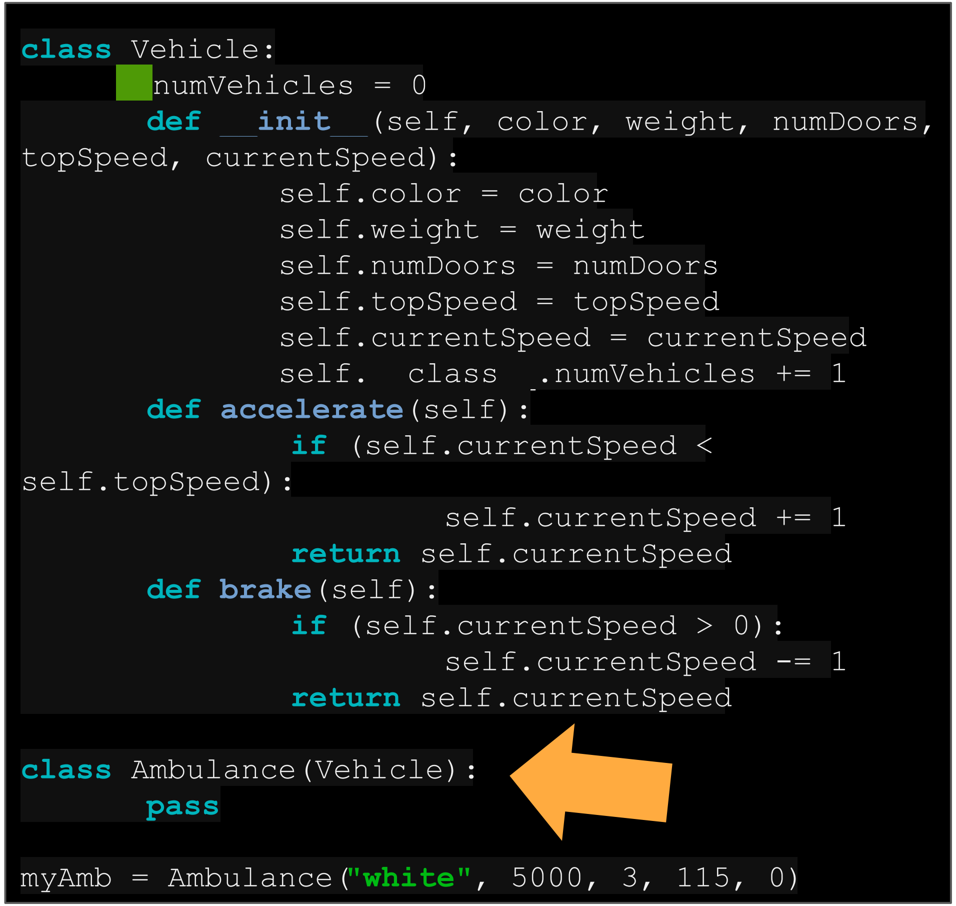


```
class Vehicle:

     numVehicles = 0

     def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
              self.color = color
              self.weight = weight
              self.numDoors = numDoors
              self.topSpeed = topSpeed
              self.currentSpeed = currentSpeed
              self.__class__.numVehicles += 1

     def accelerate(self):
               if (self.currentSpeed <
self.topSpeed):
                       self.currentSpeed += 1
     def brake(self):
               if (self.currentSpeed > 0):
                       self.currentSpeed -= 1
```

.003 Parent and Child Classes

If you think about this class from an object oriented design perspective, people rarely identify that they are driving a “vehicle.” They are more likely to say that they are driving a car, truck, ambulance, or motorbike. All are examples of “vehicles.”

The vehicle class may not be specific enough for our needs. We’ll consider our **Vehicle** class a parent class and create child classes for the specific instances of vehicle that may have properties and characteristics beyond the base vehicle class.



```
class Vehicle:
       numVehicles = 0
       def __init__(self, color, weight, numDoors,
topSpeed, currentSpeed):
              self.color = color
              self.weight = weight
              self.numDoors = numDoors
              self.topSpeed = topSpeed
              self.currentSpeed = currentSpeed
              self.__class__.numVehicles += 1
       def accelerate(self):
              if (self.currentSpeed <
self.topSpeed):
                       self.currentSpeed += 1
              return self.currentSpeed
       def brake(self):
              if (self.currentSpeed > 0):
                       self.currentSpeed -= 1
              return self.currentSpeed

class Ambulance(Vehicle):
       pass

myAmb = Ambulance("white", 5000, 3, 115, 0)
```

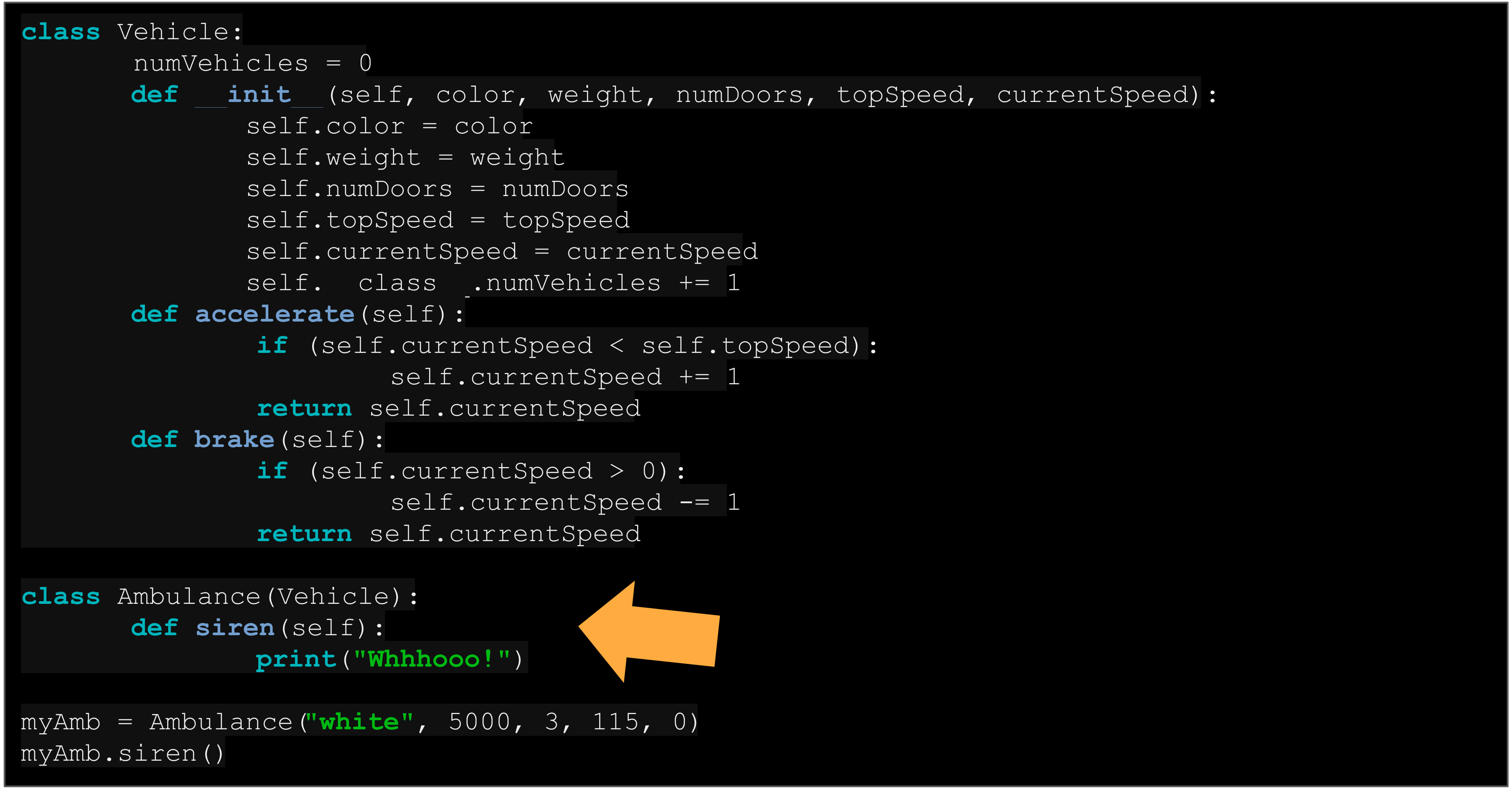
.003 Parent and Child Classes

Adding to the example, we've now created an **Ambulance** subclass of **Vehicle**. In this case, **Vehicle** is the parent class and **Ambulance** is the child class.

The instance created in the last line of the example, **myAmb**, passes the same properties to the initializer.

The Python command **pass** simply indicates that our child class inherits the same properties and methods as the parent class **Vehicle.** (Similar to how a child inherits various features from parents)

Let's make our **Ambulance** child class a bit more useful.



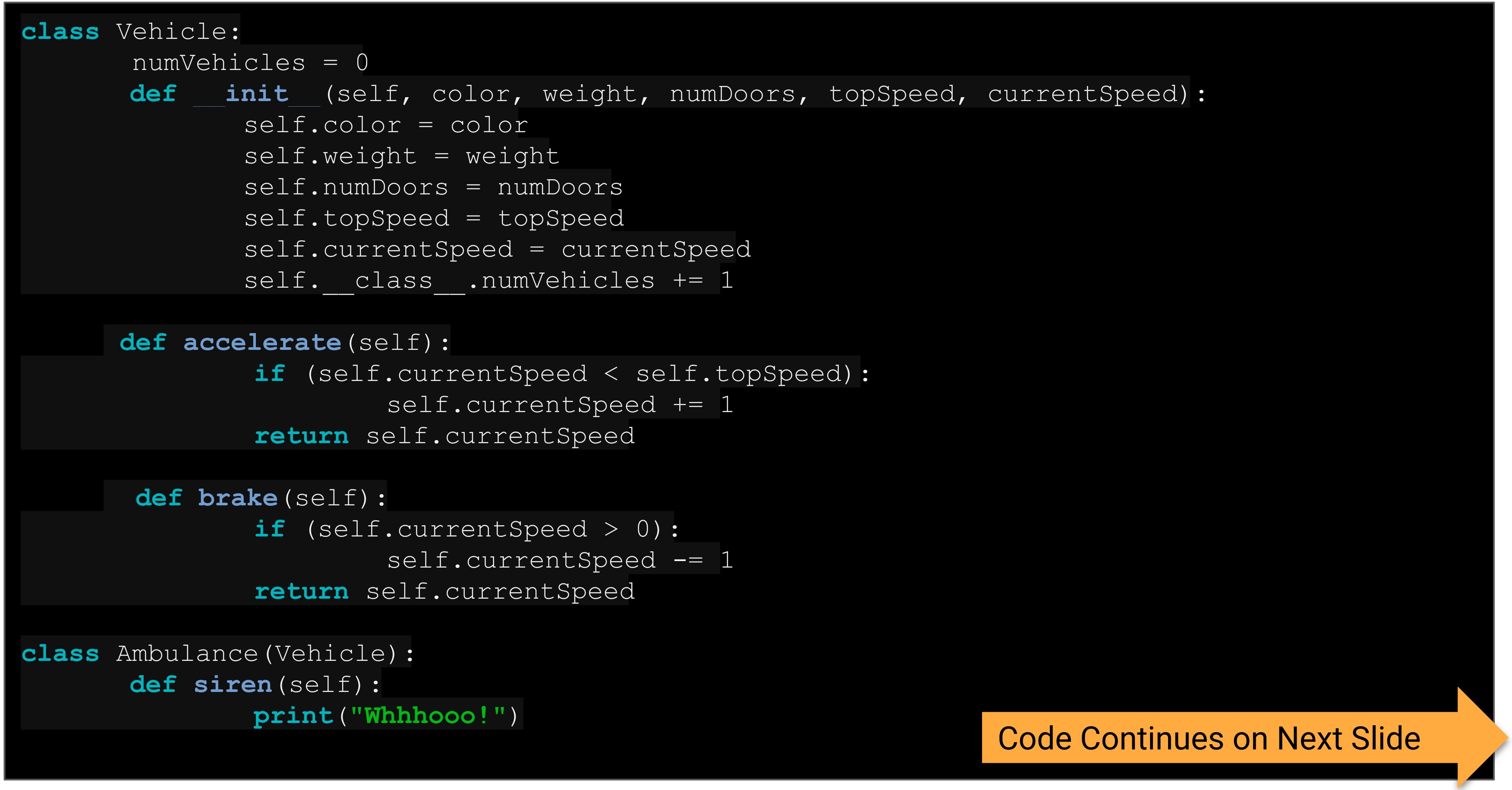
```python
class Vehicle:
    numVehicles = 0
    def __init__(self, color, weight, numDoors, topSpeed, currentSpeed):
        self.color = color
        self.weight = weight
        self.numDoors = numDoors
        self.topSpeed = topSpeed
        self.currentSpeed = currentSpeed
        self.__class__.numVehicles += 1
    def accelerate(self):
        if (self.currentSpeed < self.topSpeed):
            self.currentSpeed += 1
        return self.currentSpeed
    def brake(self):
        if (self.currentSpeed > 0):
            self.currentSpeed -= 1
        return self.currentSpeed

class Ambulance(Vehicle):
    def siren(self):
        print("Whhhooo!")

myAmb = Ambulance("white", 5000, 3, 115, 0)
myAmb.siren()
```

.003 Parent and Child Classes

Now the **Ambulance** class is inheriting all the properties methods from the parent class. We’ve removed the **pass** command, and replaced it with the **siren** method, which sounds a (not-so-realistic) siren.

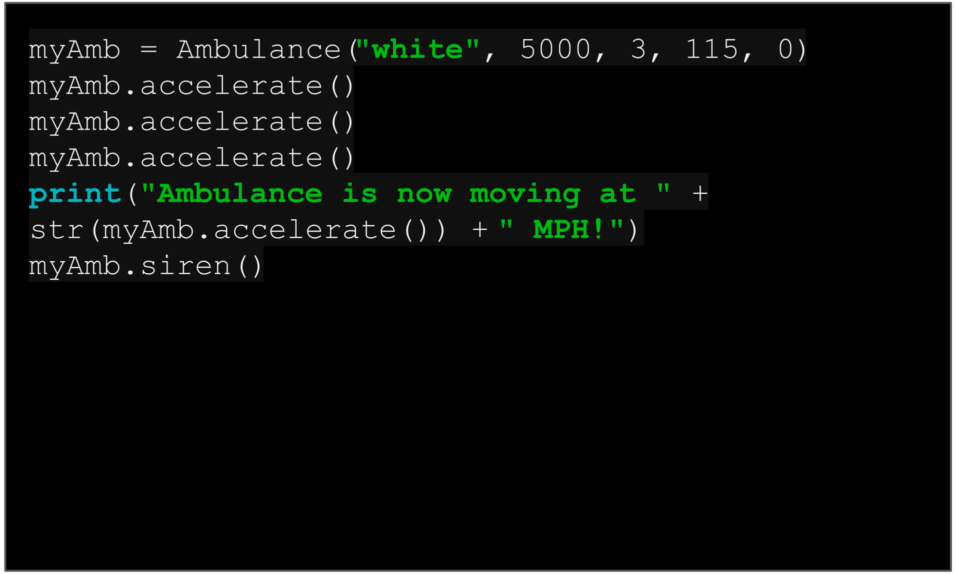
You’ll notice in the **myAmb** implementation of the **Ambulance** class, the **siren** method is used.

We can also use the methods from the properties inherited from **Vehicle**, the parent class.

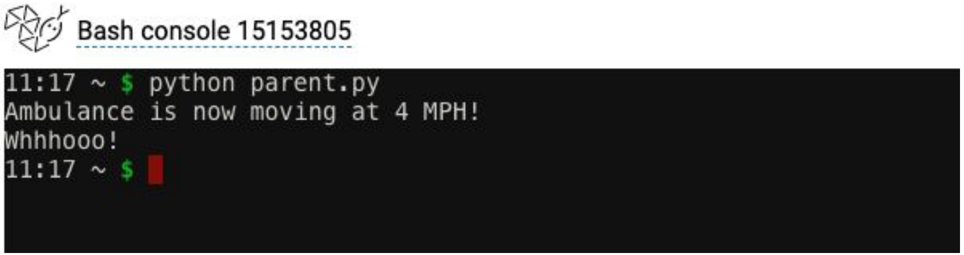


```
class Vehicle:
       numVehicles = 0
       def __init__(self, color, weight, numDoors, topSpeed, currentSpeed):
              self.color = color
              self.weight = weight
              self.numDoors = numDoors
              self.topSpeed = topSpeed
              self.currentSpeed = currentSpeed
              self.__class__.numVehicles += 1

      def accelerate(self):
               if (self.currentSpeed < self.topSpeed):
                       self.currentSpeed += 1
               return self.currentSpeed

       def brake(self):
               if (self.currentSpeed > 0):
                       self.currentSpeed -= 1
               return self.currentSpeed

class Ambulance(Vehicle):
       def siren(self):
               print("Whhhooo!")
```

Code Continues on Next Slide



```
myAmb = Ambulance("white", 5000, 3, 115, 0)
myAmb.accelerate()
myAmb.accelerate()
myAmb.accelerate()
print("Ambulance is now moving at " +
str(myAmb.accelerate()) + " MPH!")
myAmb.siren()
```



```
11:17 ~ $ python parent.py
Ambulance is now moving at 4 MPH!
Whhhooo!
11:17 ~ $
```

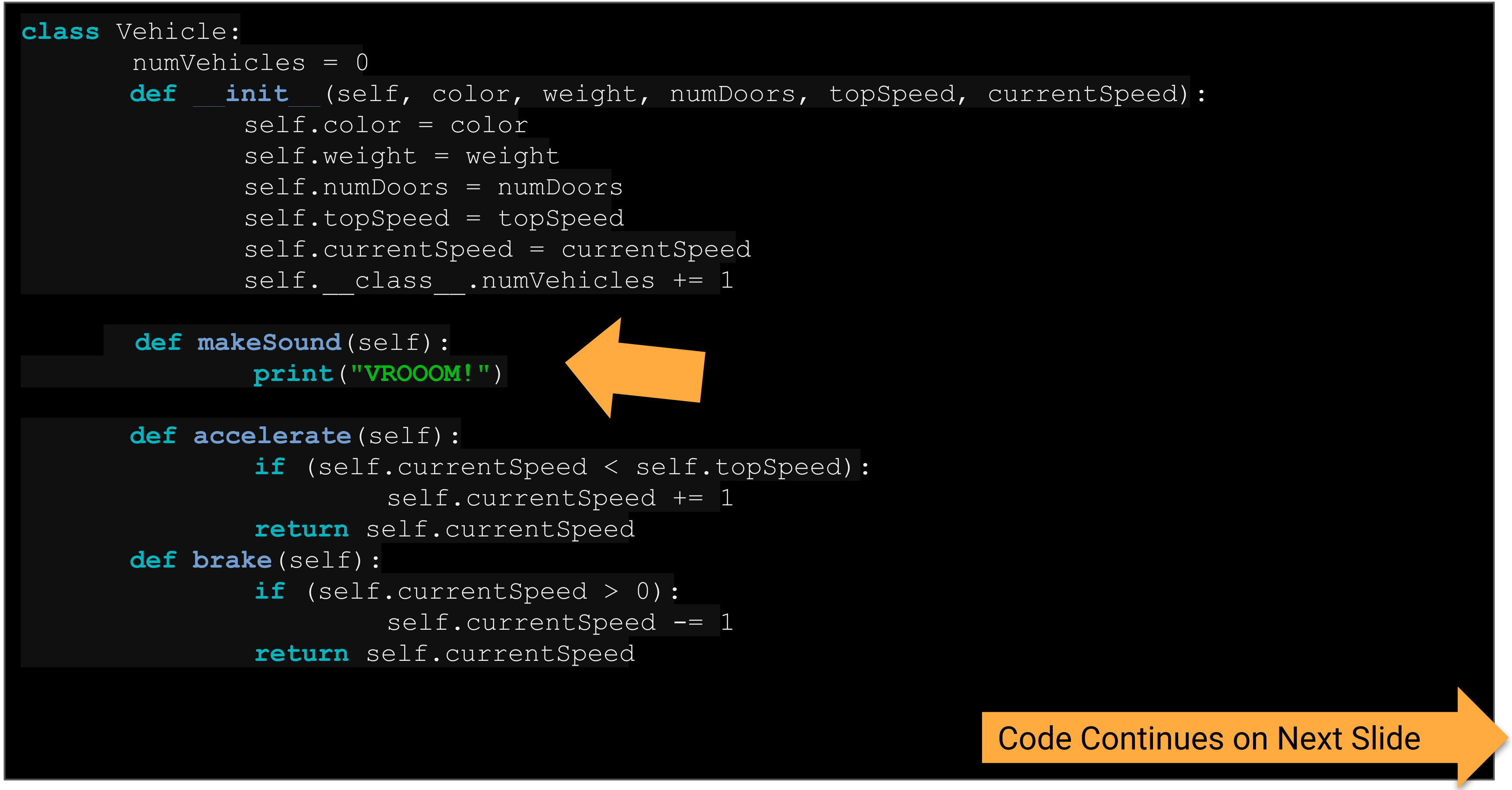
.003 Parent and Child Classes

Now things are getting interesting!

Our **Ambulance** class includes not only the **siren** method, but the **accelerate** method from the parent class **Vehicle.** Since the parent **Vehicle** uses the **currentSpeed** property in the **accelerate** method, the properties must be implemented in the child class as well!

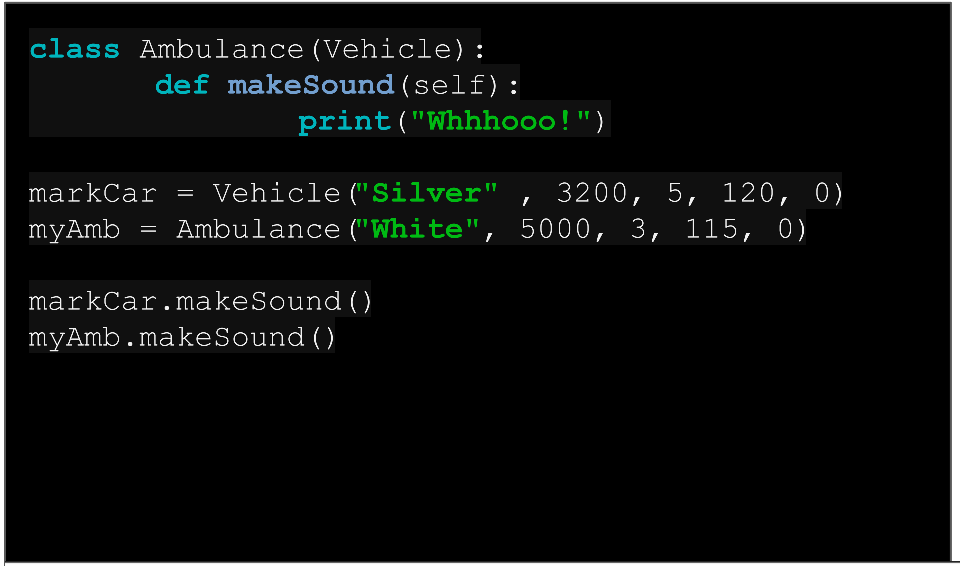
In coding terminology, we use the term **inheritance** to describe how child classes receive properties and methods from a parent.

Let's change up our implementation a bit…

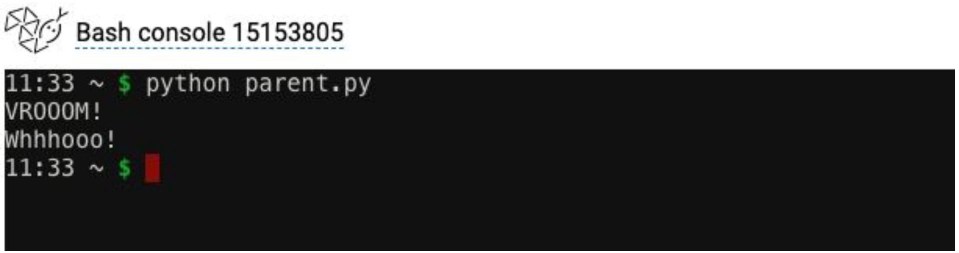


```python
class Vehicle:
    numVehicles = 0
    def __init__(self, color, weight, numDoors, topSpeed, currentSpeed):
        self.color = color
        self.weight = weight
        self.numDoors = numDoors
        self.topSpeed = topSpeed
        self.currentSpeed = currentSpeed
        self.__class__.numVehicles += 1

    def makeSound(self):
        print("VROOOM!")

    def accelerate(self):
        if (self.currentSpeed < self.topSpeed):
            self.currentSpeed += 1
        return self.currentSpeed
    def brake(self):
        if (self.currentSpeed > 0):
            self.currentSpeed -= 1
        return self.currentSpeed
```

Code Continues on Next Slide



```python
class Ambulance(Vehicle):
       def makeSound(self):
               print("Whhhooo!")

markCar = Vehicle("Silver" , 3200, 5, 120, 0)
myAmb = Ambulance("White", 5000, 3, 115, 0)

markCar.makeSound()
myAmb.makeSound()
```



```
11:33 ~ $ python parent.py
VROOOM!
Whhhooo!
11:33 ~ $
```

.004 Overriding Methods

You'll notice we've added the method **makeSound** to the parent class, **Vehicle.** Every vehicle makes some kind of sound, so this is consistent with our design.

However, in the **Ambulance** child class, the **makeSound** method is defined again, this time with a different implementation, more appropriate from an ambulance.

Child classes may **override** the implementation of a method in the parent class by redeclaring it as shown in this example.

.005 Activity 001

In this activity you'll create three classes, an **Employee** parent class, and **ptEmployee**, and **ftEmployee** child classes. The **ptEmployee** subclass should represent part-time employees and the **ftEmployee** class represents full-time employees. Every class should track the following information:

empName	String
empSocial	String
empDepartment	String

The **ptEmployee** class will also have an **hourlyPay** property. The **ftEmployee** class will have a **yearlySalary** property.

The child classes, should have a method **calculatePay** that calculates the paycheck for each employee. This works differently for each subclass. For part-time employees the number of hours worked needs to be multiplied by **hourlyPay** to come up with a payment amount. (We're ignoring taxes for simplicity's sake in this activity.)   For full-time employees in the **ftEmployee** class, **calculatePay** will divide their salary by 26 since there are 26 pay periods per year.

Create an instance of both **ftEmployee** and **ptEmployee** and implement the **calculatePay** method for both.