Subclasses

We're going to learn about subclassing (or inheritance) in Python

WE'LL COVER THE FOLLOWING

- Wrapping Up

The real power of classes becomes apparent when you get into subclasses. You may not have realized it, but we've already created a subclass when we created a class based on **object**. In other words, we subclassed **object**. Now because **object** isn't very interesting, the previous examples don't really demonstrate the power of subclassing. So let's subclass our Vehicle class and find out how all this works.

```
class Car(Vehicle):
    """
    The Car class
    """

    #--------------------------------------------------------------------
    def brake(self):
        """
        Override brake method
        """
        return "The car class is breaking slowly!"

if __name__ == "__main__":
    car = Car("yellow", 2, 4, "car")

    print(car.brake())
    # 'The car class is breaking slowly!'

    print(car.drive())
    # "I'm driving a yellow car!"
```

For this example, we subclassed our **Vehicle** class. You will notice that we didn't include an **__init__** method or a **drive** method. The reason is that when

you subclass Vehicle, you get all its attributes and methods unless you

override them. Thus you will notice that we did override the **brake** method and made it say something different from the default. The other methods we left the same. So when you tell the car to brake, it uses the original method and we learn that we're driving a yellow car. Wasn't that neat?

Using the default values of the parent class is known as **inheriting** or **inheritance**. This is a big topic in Object Oriented Programming (OOP). This is also a simple example of **polymorphism**. Polymorphic classes typically have the same interfaces (i.e. methods, attributes), but they are not aware of each other. In Python land, polymorphism isn't very rigid about making sure the interfaces are exactly the same. Instead, it follows the concept of **duck typing**. The idea of **duck typing** is that if it walks like a duck and talks like a duck, it must be a duck. So in Python, as long as the class has method names that are the same, it doesn't matter if the implementation of the methods are different.

Anyway, you really don't need to know the nitty gritty details of all that to use classes in Python. You just need to be aware of the terminology so if you want to dig deeper, you will be able to. You can find lots of good examples of Python polymorphism that will help you figure out if and how you might use that concept in your own applications.

Now, when you subclass, you can override as much or as little from the parent class as you want. If you completely override it, then you would probably be just as well off just creating a new class.

Wrapping Up

Classes are a little complicated, but they are very powerful. They allow you to use variables across methods which can make code reuse even easier. I would recommend taking a look at Python's source for some excellent examples of how classes are defined and used.

We are now at the end of Part I. Congratulations for making it this far! You now have the tools necessary to build just about anything in Python! In Part II, we will spend time learning about using some of the wonderful modules that Python includes in its distribution. This will help you to better understand the power of Python and familiarize yourself with using the Standard Library. Part II will basically be a set of tutorials to help you on your way to becoming

Part II will basically be a set of tutorials to help you on your way to becoming a great Python programmer!