CSE231 - Lab 13

Inheritance, Method Overrides

What is inheritance?

Inheritance tends to be a confusing topic to many, but if you've been following along with the concept of classes so far, it should come as pretty intuitive.

*Classes have one fundamental idea: sub (child) classes **inherit** their super (parent) class' attributes and functions.*

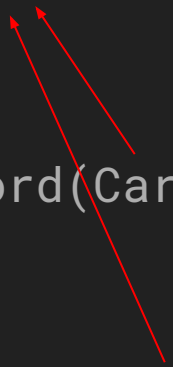
Hopefully you can see why this is useful. In the same way we package bundles of code to reuse, we can package an entire *class* to reuse.

Let's say we have a class named "`Car`", and two other classes, one named "`Ford`" and one named "`Tesla`".

The `Car` class might have attributes or functions that we want to pass down to the `Ford` and `Tesla` class, because both could be considered "cars".

Inheriting From a Parent

```
class Car(object):

    …

class Ford(Car):

    …

class Tesla(Car):

    …
```



Reusing a Constructor

Assuming our child class doesn't need a unique constructor, (we'll get to that in a bit), we can call the parent class' constructor to initialize for us.

```
class Car(object):

    def __init__(self, owner, model, plate):

        self.owner, self.model, self.plate = owner, model, plate

class Ford(Car):

    def __init__(self, owner, model, plate):

        Card.__init__(self, owner, model, plate)
```

What is this?

```
Card.__init__(self, owner, model, plate)
```

It's some funky syntax, right? We're effectively saying:

1. Use `Card`'s constructor ... `Card.__init__`
2. With `Ford`'s class ... `(self,`
3. And pass `Ford`'s constructor arguments ... `owner, model, plate)`

This is called an *unbound call*. There is an equivalent way of doing this that you might prefer, albeit a bit more ambiguous than the one above.

```
super().__init__(owner, model, plate)
```

Sidenote

I think it's important to highlight the difference between this:

```
ClassName()
```

And this:

```
ClassName
```

This also applies to function names, too. When you add those parentheses, you are *calling* that class -- you're *instantiating* it. When you use the name *alone*, you are accessing the class' reference. You're taking the class and accessing its member *without* instantiating it. There's much more on this topic that we could get into, but this is as far as we'll go in this course. (Look into "*decorators*" if you're interested)

Example Time

So all of this was just about *how* you sub-class. But now, let me show you why creating hierarchical structures like this can be useful.

I'll be showing this on my code editor since PythonTutor can get a bit cluttered with class definitions.

Method Overriding

When creating sub-classes, there are times where you might want to have the same function as the super-class, but slightly altered to cooperate with the sub-class.

We can achieve this via *method overriding*, essentially re-defining a method function, with the same name, but with a different algorithm. You can do this for *all methods*, including the constructor and magic methods.

And that’s it!

Congratulations! You’ve just made it through the hardest part of the course!

As I said, there *are* more things that you can do with classes (and Python in general) that are more computer science-y and not well suited to most engineering problems.

Classes may appear to be a uselessly complicated way to store things, but after you program for a while, you’ll realize how much utility and convenience they offer.

From this point on, you’re either about to take the final exam (Fall semester), or you’ll have some bonus topics (Spring semester) for a couple weeks.