Classes

Steven Walton
PS 232 Computational Methods
Department of Physics
Embry-Riddle Aeronautical University
Prescott, AZ 86301

December 3, 2014

Why do we code?

Seriously, why do we code? It's because we're lazy. No, I'm not joking. Look at the things we have done in class so far. Do you want to do that by hand? Pen and paper? Could you do it by pen and paper? I'm guessing you're saying to yourself that the answer is no, and rightly so. Already we have learned about loops and definitions. They make our life drastically easier. For repetitive tasks computers do things much faster than we could ever imagine to do by hand. And this is why we code. Because we are lazy.

Classes

So you may have started to think to yourself at this by the time you may have started asking yourself what we would do if we wanted to use a bunch of definitions over and over again. What if we wanted to create a database of lectures, astronomical data, friends, or whatever. How would we do this? The answer is actually in classes. Let's do a simple example.

```
class Drink:
def __init__(self,drinkType,cost):
   self.drinkType = drinkType
   self.cost = cost

description = "Nobody described this wonderful drink, we'll never
              know..."
creator = "Nobody staked their claim to this drink, I guess it's
           mine now."

def drinkType(self):
    return self.drinkType

def cost(self):
     return self.cost
```

```
    def describe(self,text):
        self.description = text

    def creatorName(self,text):
        self.creator = text
```

What we have done now is create a class for drinks. This is great if we want to have a bunch of different drinks in our code. No longer do we need to write these definitions over and over again, but we can store them with classes. So let's play with these classes a little. We have created the code, but we have yet to actually make a drink.

```
espresso = Drink("Coffee",1.25)
few = Drink("Whiskey",4.00)
```

We have now created a drink, but all we have is what type of drink it is and the cost of it. If we ask for "espresso.creator" we get back the default value, and the same thing happens if we ask for the description. So let's modify this a little bit.

```
espresso.describe("Elixir of life")
espresso.creatorName("Becca")
few.describe("An amazing rye whiskey)
few.creatorName("Steven")
```

Now if we print "few.description" or "espresso.creator" we get back the values that we gave. Note that we have a default value and that the definitions are not the same as these values. So that we have a definition to define the value and the value itself. If we did not do it this way we would not be able to correctly create and store our drinks.
Let's say that we want to store these. If you remember back to an earlier lecture we talked about dictionaries. So let's store them that way.

```
dictionary = {}
dictionary["Few Whiskey"] = Drink("Whiskey",4.00)
dictionary["Few Whiskey"].authorName("Steven")
dictionary["Espresso"] = Drink ("Coffee",1.25)
dictionary["Espresso"].describe("The thing that keeps Becca alive")
```

Now we have these drinks stored in a dictionary. We can access them like "dictionary["Espresso"].description". As we have discussed before we can store this into a json format or you may be interested in the base module called pickle.
Classes have a lot of advantages over what we have previously used. A big advantage is security. It protects against errors in code. It also makes code reusable and it separates low level operations from high level operations. This is extremely useful because it hides stuff from the user. This means that the user does not need to interact with the back end of the program and this makes it easier for users to interact with programs.
One other thing to note is that we can add and remove attributes to our classes. So we can do "dictionary["Few Whiskey"].year = 2001" so that we can know what year the whiskey is (as all whiskey drinkers know, this is important). We are also able to delete attributes with the "del" command.
I have created a small program to take orders of drinks. This would make for a poor cash register system

though. The reasons being that if we have a real cash register we will only want to place orders for things we have in stock. And if they are in stock we will know the price already. Another thing is that it asks if you would like to add another drink to the order each time. We don't want to do this. We want it to quit when we are done and just keep taking orders till then. I hope by now you know how to fix this problem. If you are feeling adventurous then make a GUI program that also has a small description of what the drink is. We will also want to destroy the orders that we had previously and restart the entire program for the next customer. This is common when using classes and you should keep it in mind. Especially if you are using code that will continuously run.

Inheritance

Many times we want to have a parent and child class. The child class will inherit attributes from the parent class. So let's make an example with parents and children.

```
class Parent():
    parentAttr = 100

    def __init__(self):
        print "Parent Constructor"

    def parentMethod(self):
        print "Parent Method"

    def setAttribute(self, attr):
        Parent.parentAttr = attr

    def getAttribute(self):
        print Parent.parentAttr

class Child(Parent):
    def __init__(self):
        print "Child Constructor"

    def childMethod(self):
        print "Child Method"
```

Now we have a lot of useful things from here. If we run

```
c = Child()
c.childMethod()
c.parentMethod()
c.setAttribute(150)
c.getAttribute
```

We will end up getting back

```
Child Method
Parent Method
150
```

As you can see the child class inherits all the attributes from the parent class. This is an extremely useful process and can keep your code a lot more organize. This is the power of object oriented programming. We are much better able to organize our data and routines. With it comes better speed in both writing and running. There are a lot of advantages to classes, but this is only an introduction. Please read more about the classes to get a better understanding. The best way to understand code and its capabilities, though, is to practice.