{:title “Python - Classes” :layout :post :date “2016-10-15” :tags [“Python” “Code” “Guide”]}

Creating classes is the bread and butter of object oriented programming. Before diving stright into the good stuff lets review and cover some more conceptual ideas before going into programming. The Python syntax for creating classes is pretty stright forward. Specially since creating a class in python has very little syntax then some of the other programming languages. We have seen different types so far in Python such as Strings/Integers/Floats. However what if you wanted to make a type? Thats what classes are actually. In a way they are user defined types. Scoping and Namespaces. Scoping is the idea that certain variables are defined in certain contexts. In python objects have attributes that are specific to that object. Instances which is what is created from a class share class variable shared between all instances of a particular class.

```
class Person:

    population = 0

    def __init__(self, name, money):
        self.name = name
        self.money = money



def main():

    a = Person("Batman", 999999999)

    print(a.name)
    print(a.money)
    print(a.population)


main()
```

So the syntax for creating a class is by using “class” all lower case. Followed by what you want to call the class in this case the class is called Person. The first letter is capitalized when giving the name of a class and ending with the double colon :. Also the init has two underscores infront of it and than followed by two underscores. When you run this this is the result you get.

```
Batman
999999999
0
```

Explanations below Line 1:

```
class Person:
```

This is the syntax in which you begin when defining a class. In this case our class is called Person. As I said before the name of the class must be capitalized. It is similiar to defining a function as everything under it is indented by a tab or spaces but just stick to one. There is a whole debate regarding tabs vs spaces but just stick to one and don't intermix spaces and tabs and you should be fine. Line 3:

```
population = 0
```

This in python is known as a class variable. This is shared with all other "instances" that you create that is related to the class Person. Instances which you'll see are created using a similar syntax as a function but binding it to a symbol or variable which you saw in the main function. Line 4-7:

```
def __init__(self, name, money):
        self.name = name
        self.money = money
```

This may be a little frightening to see. But this is not too different from making a function. *__init__* is special that it is called whenever a class instance is created using a constructor. Here you are telling python that the constructor function must take 2 arguments. But wait what is this self argument that the constructor is taking? ___init___ always takes self as the first argument and the rest of the arguments can be applied as attributes like we are doing here. Attributes are called using the . syntax on the variable that is holding the instance. Line 14:

```
a = Person("Batman", 999999999)
```

There is a lot packed into here that doesn't meet the eye. So I will try my best to not leave anything out and explain it the best I can so that this makes sense and there is no confusion since it is very easy to get confused here. You can consider Person("Batman", 999999999) as "making" the instance of the class Person. This is what is known as a constructor. It is creating an instance of the class Person based on the specifications that you set in the def init. The constructor will take two arguments a name and the money that the person has. Note that you don't pass three arguments but only two this is because self is a special argument that gets passed onto itself already that you don't have to worry about hence self. Line 15-17:

```
print(a.name)
print(a.money)
print(a.population)
```

So to get the class variables and attributes you use the variable you attached to it and use the . syntax to get the attribute of the class. name and money are attributes that are specific to the instance created. However population is a class variable that is shared with all instances. To demonstrate this point add the following to your main function below all the prints.

```
    k = Person("Superman", 100000)

    print(k.name)
    print(k.money)
    print(k.population)
```

Now running this you can see that k as the attribute population as well. This attribute is the same attribute as the attribute from a. This is shared between both superman instance and batman instance. You can change the attribute and it won't be updated for the other like the following. Add the following in the body of your def main. Keeping the rest the same.

```
a.population = 3
print(a.population)
print(k.population)
```

So your main function should look like this now.

```
def main():

    a = Person("Batman", 999999999)

    print(a.name)
    print(a.money)
    print(a.population)

    k = Person("Superman", 100000)

    print(k.name)
    print(k.money)
    print(k.population)

    a.population = 3
    print(a.population)
    print(k.population)
```

This is the result:

```
Batman
999999999
1
Superman
100000
1
3
1
```

Now running this will result seeing that the a.population results in 3 while the k.population is still 1. So you can change an attribute using the dot syntax and the attribute name of the instance followed by the assignment operator to set the attribute for that specific instance to a different value just like you do for other variables expect it will be specific for that instance alone. Lets look at another feature of classes. So lets change our entire python file to this.

```
class Person:

    population = 0

    def __init__(self, name, money):
        self.name = name
        self.money = money

    def getname(self):
        print(self.name)

    def getmoney(self):
        print(self.money)

    def getpopulation(self):
        print(self.population)


def main():

    a = Person("Batman", 999999999)
    k = Person("Superman", 100000)

    print("Batman info:")
    a.getname()
    a.getmoney()
    a.getpopulation()
    print("Superman info:")
    k.getname()
    k.getmoney()
    k.getpopulation()

main()
```

Results:

```
Batman info:
Batman
999999999
0
Superman info:
Superman
100000
0
```

Notice that we have more defs within our class now. We have getname, getmoney and getpopulation. These are functions specific to the class Person. So that means that these functions will only work on instances that are from the class Person. now you see something you might have remembered doing previously when working with numbers.

```
print("Batman info:")
a.getname()
a.getmoney()
a.getpopulation()

print("Superman info:")
k.getname()
k.getmoney()
k.getpopulation()
```

Now you notice we have parantheses at the end to call the functions for that instance. This is the syntax to call a function from a particular instance of a class. However if you use just getname() it won't work because it is specific to the class Person and so you have to use it with an instance of class Person. This is very much like doing Math.cos() expect instead of calling the function from a specific module that is avaliable you are calling a function from a class. More specifically an instance that has the functions from the class. Wrapping up:

Lets review because there is quite a bit that was talked about here but maybe not fully discussed about. First we have the whole concept of classes. For now classes can be thought of has a seperate namespace that you have with variables in that namespace. Attributes are very much like variables that are specific to the instance. Instances being what is created when using a constructor which is how we created the Batman and Superman instances from the class Person. The constructor looks for a template on how to make the instance and then creates it using the "**init**" function that is called when an instance is made. The init function should not be treated like a normal function within the class. It is ONLY called when you create an instance such as when batman and superman was first created by calling the constructor. The constructor calls **init** if there is one and that is the only way that function gets called. Until you make another instance that is then it is called again for that instance. I threw around a lot of words and it may still not click much for you what exactly instances, classes, constructors are exactly. However it just takes time to get used to seeing them and practice making your own classes and playing around with them to figure what what they can be or can't be. I didnt go over too well what exactly the a and k are doing but they are essentially objects holding the instance. You can check this using type on both a and k by adding the following lines at the end of the main function.

```
print(type(a))
print(type(k))
```

You'll see from the result that both a and k are of class Person. Now that you have created classes you can greatly abstract your programs with all kinds of self created classes. However be weary that not all cases call for classes. You can sometimes just write a function instead of actually make an entire class for it. Knowing the difference between when to create a function or a class can sometimes be tricky but generally you should try to create functions. The only time you need to create a class if you need to create objects that require specific individual attributes and need to be manipulated in a special way that a function would constrict you from doing.