Python

Methodologies for Data Science

Lars Sorensen - 16:137:603
Fall 2015 - biglars@cs.rutgers.edu
Unit Five

Python Methodologies for Data Sciences

The list is now two slides.... Look at all you've done

Loading Python on your machine - Figuring out how to run IDLE - Using the Python interactive command window

Creating programs - Print function - Variables - Assignment - Arithmetic

Modulus - Strings - Integers - Floats - Casting - Input function

Booleans - Arithmetic conditionals (>, <, ==, !=, <=, >=) - Boolean operators (And, Or, Not) - if - elif - else

Sequences - strings - slicing - indexing, reverse indexing - len function - ord, chr - string methods

Lists - split method - append method - mutability of lists - mixed data types of lists

Loops - Iteration - for loops - range function - nested loops - while loops - sentinel loops - counters - accumulators

Python Methodologies for Data Sciences

Interactive loops - definite loops - indefinite loops - keyword ‘in’ as set conditional

Functions - arguments - return types - scope - File I/O - opening, closing, reading from files, writing output to files, pickling

importing libraries - the Python Standard library - writing your own library - from import - third party modules, etc

Sorting and searching, binary search, linear search, Big O, Working with Third party libraries, PIP, matplotlib, Dictionaries, Exception Handling…

You’ve done all of this in only two months… It’s a good job

Python Methodologies for Data Sciences

Welcome, my brave Python soldiers, to Unit 5. This is Unit where we shake things up a bit. No more learning syntax or playing around with Computer Science, in this Unit we're going to show you an entirely new way to write your programs! Now, don't freak out. It's not hard to understand and many find it very intuitive. Object Orientation is just a new way to think about how we represent our data. What if, instead of abstracting our data and operations to a higher level we came down a step and actually tried to model the objects we are working with?

For example, what if I want to answer questions about a billiard balls on a table? Would that be easy if I were creating a program in the imperative style that we just learned? Wouldn't it be easier to simulate the actual object itself, to describe a ball and what it can and can't do and then just have fifteen of them interact?

Object Orientation or Object Oriented Programming (OOP) is a domain rich with buzzwords and bizarre sounding topics. Inheritance, Polymorphism, Deconstructors, Overriding Functions, all of these strange terms will be explained and understood. OOP is a fairly recent phenomena. Having come from academia and the PARC labs in the sixties, it was brought to most of our attention with a language called Smalltalk in the early eighties (I was actually alive then! Scary. MTV actually played music back then) and became very popular with C++. Once Java hit the scene (mid nineties) OOP was here to stay.   Let's take a look...

What is Object Orientation?

Python Methodologies for Data Sciences

Object Oriented Programming

So far, we have been writing programs in a way called Imperative Programming. It's like a recipe.

We have our data (ingredients) and then we do things to the data (like mixing, stirring, baking) and then we have a result.

But this is not the only way to write a program...

Python Methodologies for Data Sciences

As we've said, so far we have been writing programs that are like grocery lists, or to-do lists. Just lists of things to do. We do them, sequentially, and arrive at the solution to a problem in this way.

Another way of creating programs is with object oriented programming. This way of writing programs lets you create "objects" and you solve problems by using these objects and letting them interact with each other.

Python Methodologies for Data Sciences

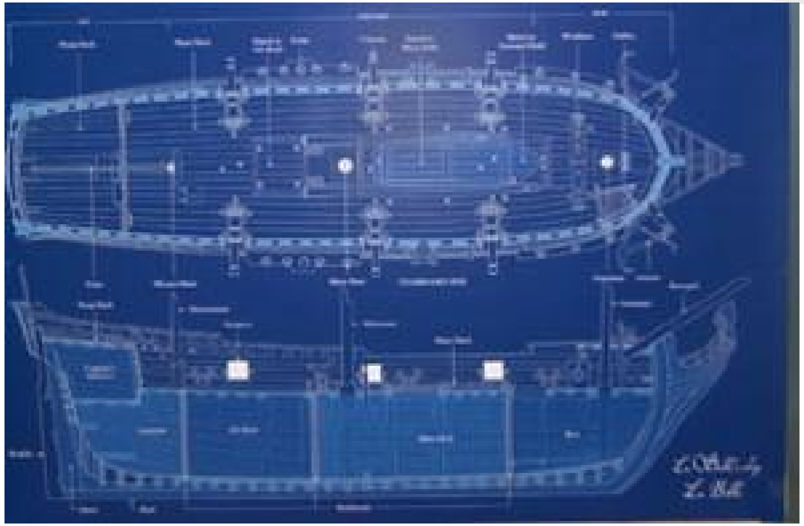
I mean, what is an object? What can we say about them?

They are collections of related information that have a certain set of operations that they perform, yes? If this is the case shouldn't we keep the data and the operations that work on that data together?

Python Methodologies for Data Sciences

Here's another important aspect. Your particular object, whatever you want to model and use in a computer program, is an object that has common global attributes and behaviors with other objects like it. Birds have wings and beaks and feathers, from ostriches to swallows. And just like that, an object is a grouping of descriptions and actions.
What if we thought of an object like an "instantiated" case of a bigger blueprint, a recipe for making this particular object.

Python Methodologies for Data Sciences



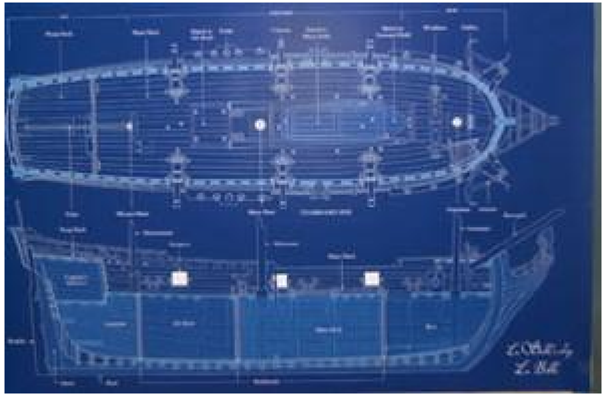
In OOP we call these blueprints classes. Stay with me, you'll get it in a second...

Classes

Python Methodologies for Data Sciences

The foundation of object oriented programming is the creation of classes. They are merely blueprints that let us create or “instantiate” objects that we can then use in our programs.

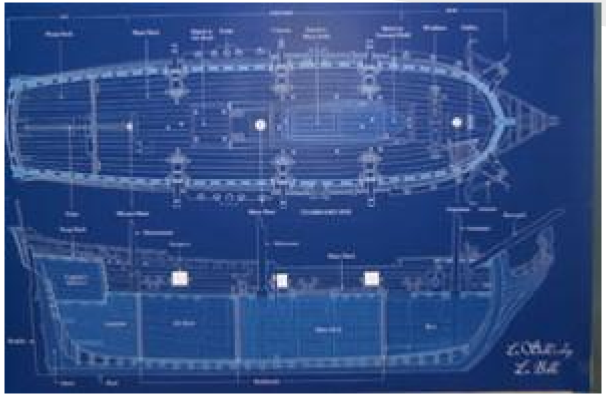
Python Methodologies for Data Sciences



Like a regular blueprint, a class can be used to describe the basics of an item, its attributes and its behaviors.

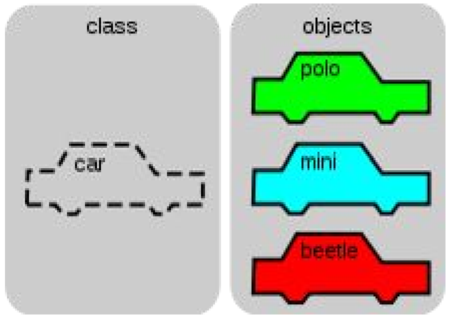
But this does not mean that the resulting objects have to be the same...

Python Methodologies for Data Sciences



A blueprint can create a wide variety of items, yes? Classes are the same way.

Python Methodologies for Data Sciences

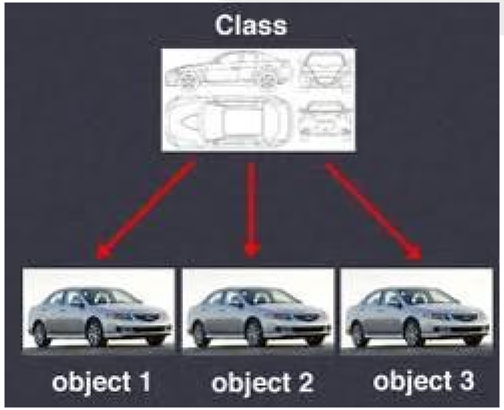


The structure of a class is easy.

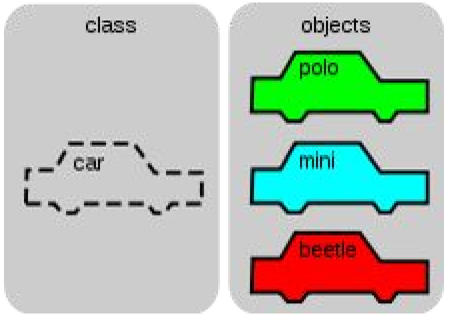
You have DATA and you have FUNCTIONS.

For a reason I still have not discovered, we no longer call functions functions when we deal with them in OOP, we call them METHODS.

When you hear the word METHOD just think "That's a function that's in a class."



Python Methodologies for Data Sciences

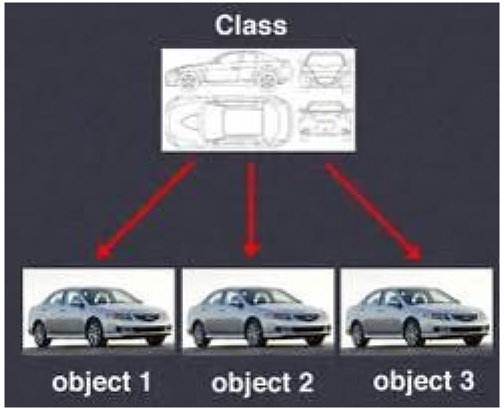


Data

- Attributes

Methods

- Behaviors and Actions



Python Methodologies for Data Sciences

As usual, most of this sounds nuts until we see an example and then you do a

"OOHHHH, Now I get it..."

So let's go get it...

Python Methodologies for Data Sciences

A die class....

A die class....

What would a die have?

What kind of attributes?

Well, it would have a number of sides.
It would also have a value, whatever is showing....

What kind of behaviors?

You can roll it.
You would also need a way to see its value, to “get” it.

Can we do this?

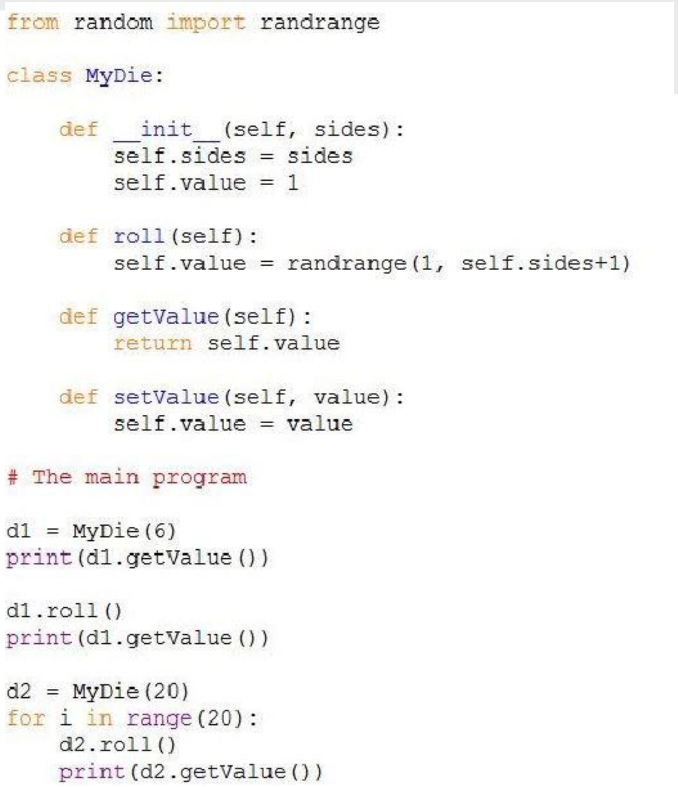
Python Methodologies for Data Sciences

To your right is a program that defines and uses a class for a die, not dice, a die. We will pick this piece of code apart and puzzle it all out.

First, we use the keyword "class." See IDLE make it orange? It's a keyword after all. Then we give the class a name. We use the same conventions we use for variable and function names when we name classes.

See that first func- I mean method? The one called __init__? (That's 2 underscores, init, and two more underscores; us Pythonheads call these double underscores "Dunders.")

We need to discuss this guy in detail...

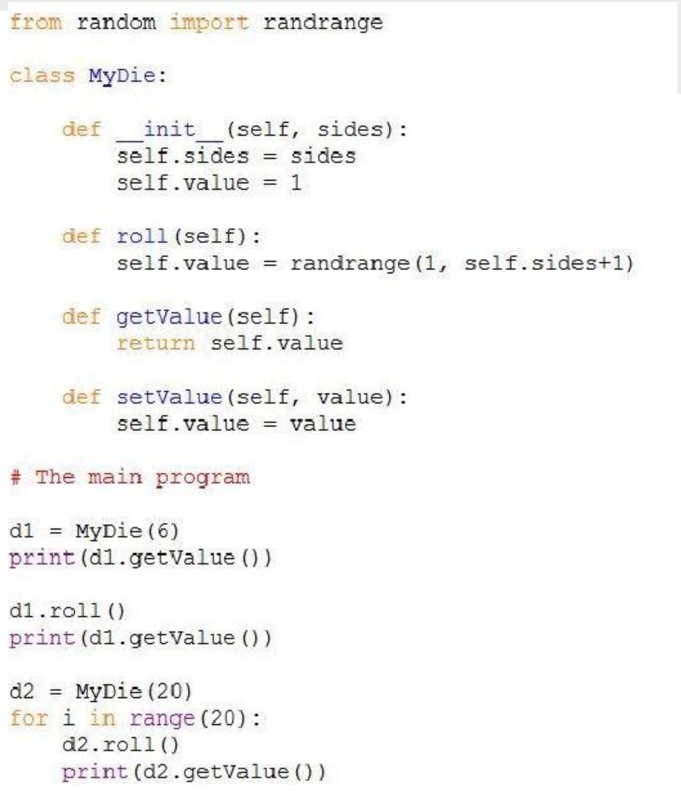


```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

Constructors

When we instantiate or get a real working copy of our class we are going to need a method that does all of the things we want done when the object is created or "constructed." We call this initial method a "Constructor." In Python the constructor always has the name __init__. It takes a variable called self. Self is a special variable name used in classes. There's no way to know what your variable will be called when you create a class, so self is a nifty way for a class to refer to itself, or at least an object that was created with this particular class.

Ever program with Java or C++? Self is like "this" in those languages. Note that when defining methods, you must pass self in. However, when calling your methods, self will be passed in implicitly.

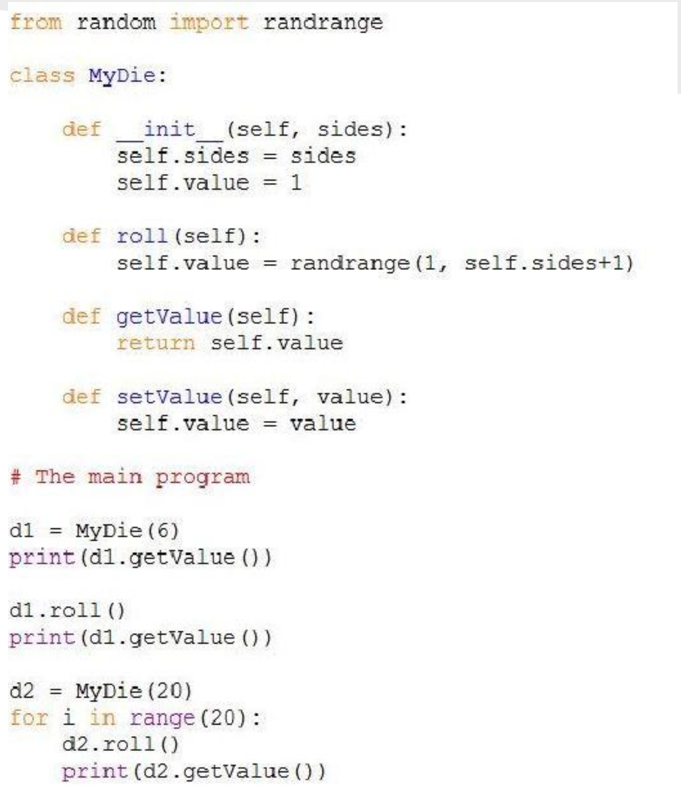


```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

Constructors

Now, constructors are very important because they handle the instance data. The data that is unique to your object is assigned in the constructor. As we can see, the MyDie class has two instance variables for data. One we pass into it as we instantiate it by calling the constructor (d1 = MyDie(6)) and one, self.value, that we set to 1 as a default every time we create an instance of MyDie.

We set them both with the self.<variable> naming schema. We will see why this is important later, for now just know that each object will have it's own data and we want to keep it unique, so we relate it to the object's variable name. This is called the object's "instance data."

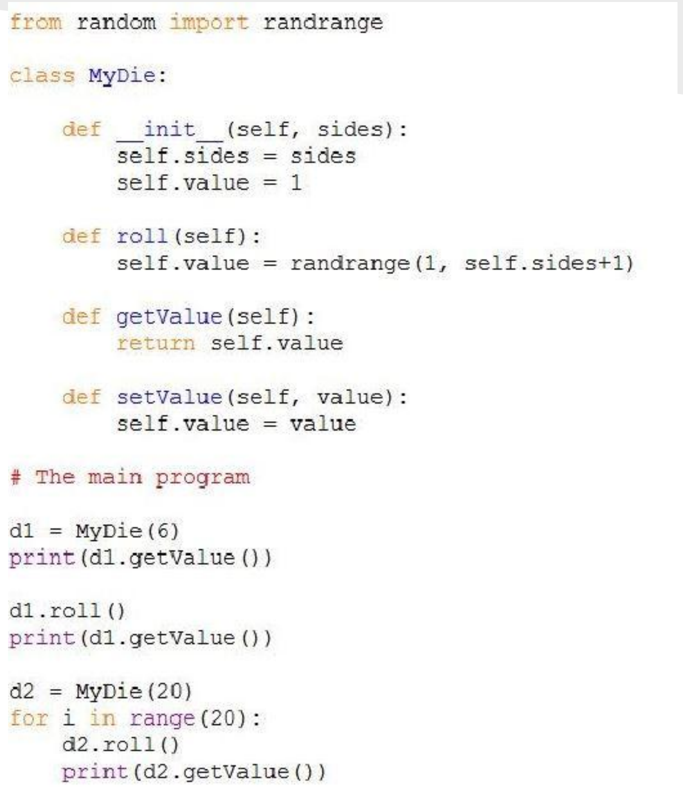


```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

Methods

After the constructor sets the data (it can do more but our simple example just needs the data set) we begin to define the other behaviors of our object.

It’s a die so we have to roll it to change its value, yes? So we create a roll method. This method uses a function called randrange to generate a random number between 1 and sides+1 (randrange, like range, is exclusive at the end of the range so sides+1 is the first thing we don’t want.)



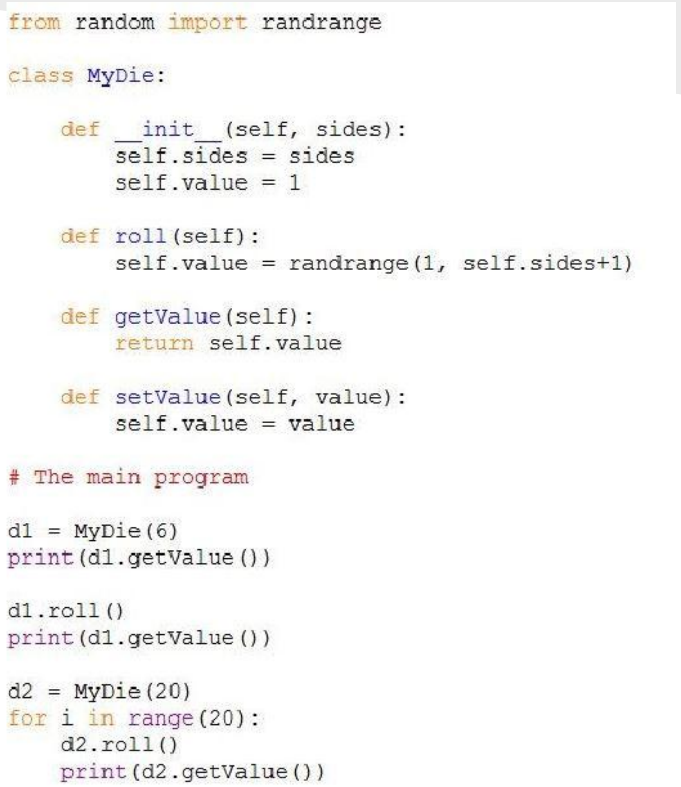
```python
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

Python Methodologies for Data Sciences

Getters and Setters

Lastly, we deal with our "Getters" and "Setters." These are methods that allow us to get data from our object and to set the variables to new values. Usually, each variable in a class has a get<variable> method and a set<variable> method.

Here we have getValue() and setValue()



```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

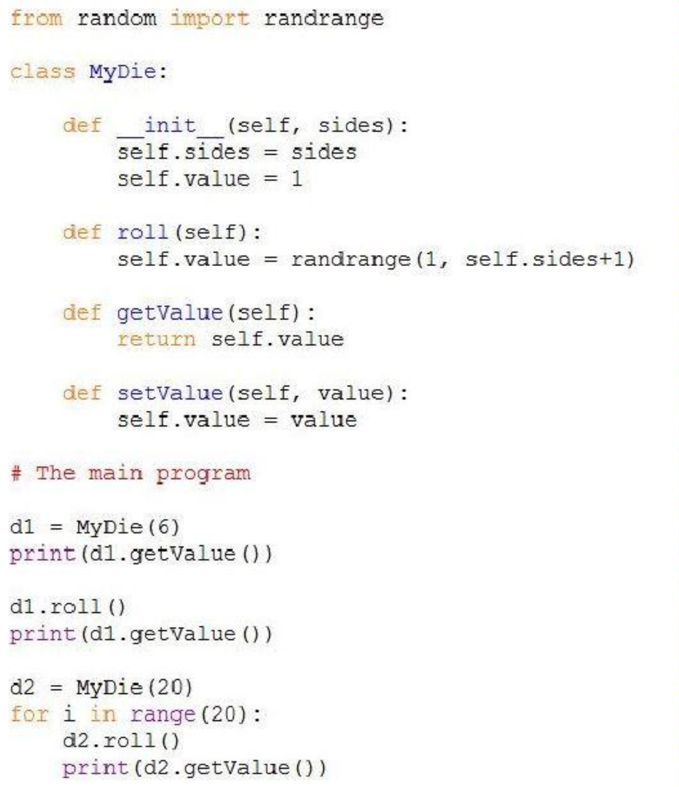
Python Methodologies for Data Sciences

Enough already, let's see it in action. Here's the run...

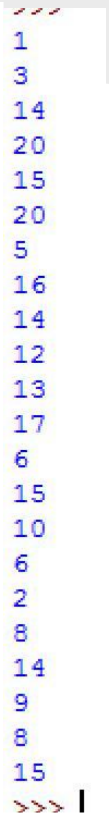
Notice how the class is defined at the top of the program. Classes are like functions this way, they need to be defined in the code before you use them.

This is going to make a great deal more sense when I trace through it on YouTube, but the basics are this. I created a die called d1. I made it 6 sided. I printed its initial value and it was 1, as that's the default. I accessed that data with the getValue method.

Then I roll it (d1.roll()). After I roll it I print out the value again. This time it's a 3. Those are the first two lines of the output.



```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```



```
1
3
14
20
15
20
5
16
14
12
13
17
6
15
10
6
2
8
14
9
8
15
>>> 
```

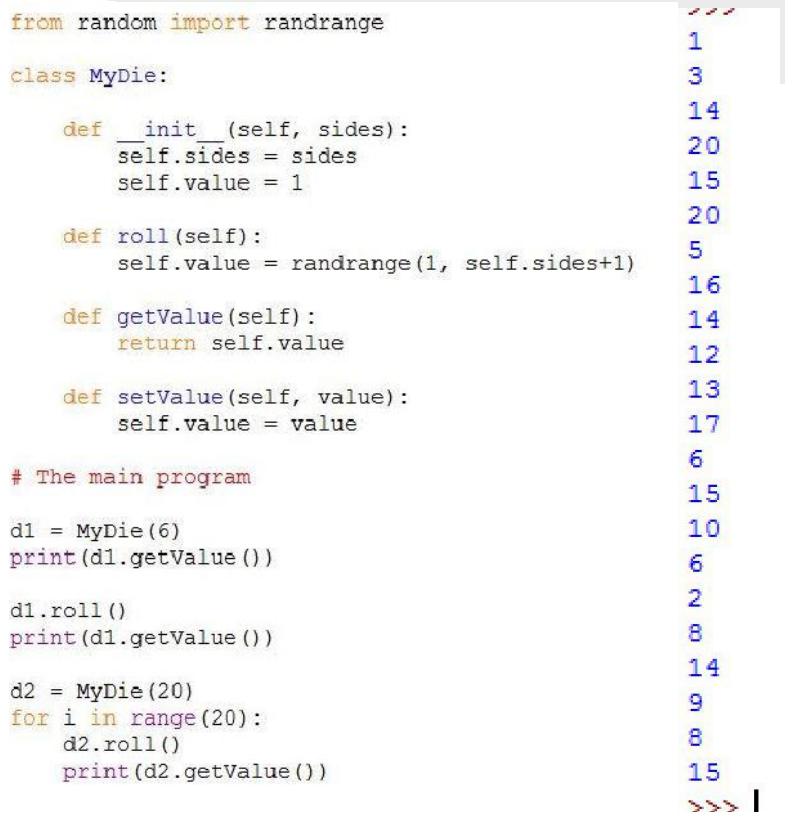
Python Methodologies for Data Sciences

Enough Already, let’s see it in action. Here’s the run...

The next twenty lines are me rolling a twenty sided die.

I create the die <--- d2 = MyDie(20)
I go into a loop
I roll the die <---- d2.roll()
and then I print the value <---- print(d2.getValue())

That’s it. I created a die and I am using it in a program.



```
from random import randrange

class MyDie:

    def __init__(self, sides):
        self.sides = sides
        self.value = 1

    def roll(self):
        self.value = randrange(1, self.sides+1)

    def getValue(self):
        return self.value

    def setValue(self, value):
        self.value = value

# The main program

d1 = MyDie(6)
print(d1.getValue())

d1.roll()
print(d1.getValue())

d2 = MyDie(20)
for i in range(20):
    d2.roll()
    print(d2.getValue())
```

```
>>>
1
3
14
20
15
20
5
16
14
12
13
17
6
15
10
6
2
8
14
9
8
15
>>>
```

Python Methodologies for Data Sciences

Python Methodologies for Data Sciences

Now that you've seen an object in action you may have realized something. You've been using them all along.

list is just a class. When you say a = [2,3,4] you are creating a list object.

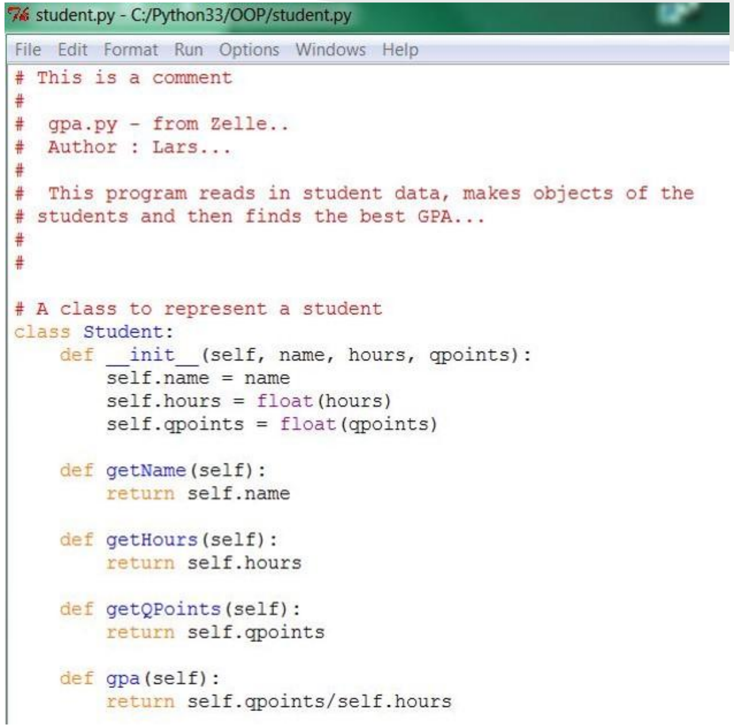
After you create or "instantiate" it, you can run methods on it:

a.replace(2,7)
a.sort()

These are just methods for the object defined in the list class, that's all...

Python Methodologies for Data Sciences

We do not have to have classes and objects mimic concrete real world things all the time. We can also represent abstract things, like a college student.



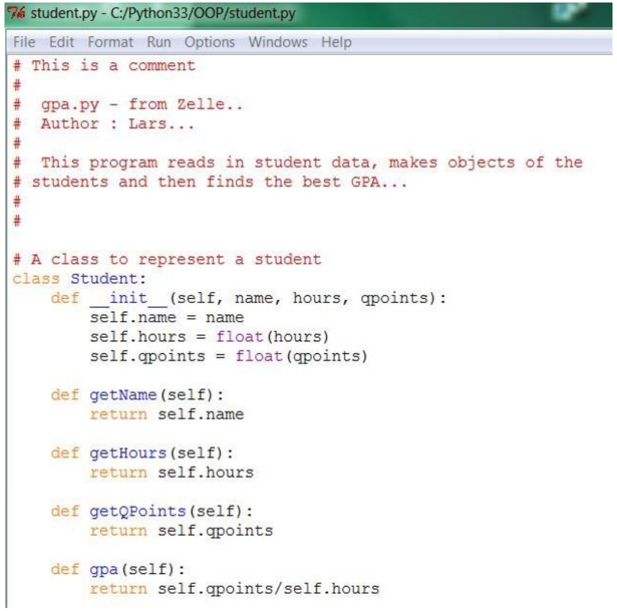
student.py - C:/Python33/OOP/student.py
File Edit Format Run Options Windows Help

```
# This is a comment
#
#  gpa.py - from Zelle..
#  Author : Lars...
#
#  This program reads in student data, makes objects of the
# students and then finds the best GPA...
#
#


# A class to represent a student
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)

    def getName(self):
        return self.name

    def getHours(self):
        return self.hours

    def getQPoints(self):
        return self.qpoints

    def gpa(self):
        return self.qpoints/self.hours
```

Python Methodologies for Data Sciences

This is the code from pg 311 of Zelle, the Data Processing with Class example.

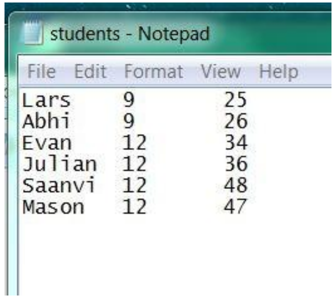
This is a more practical example of how object oriented code would look in a real world situation.


student.py - C:/Python33/OOP/student.py
File Edit Format Run Options Windows Help

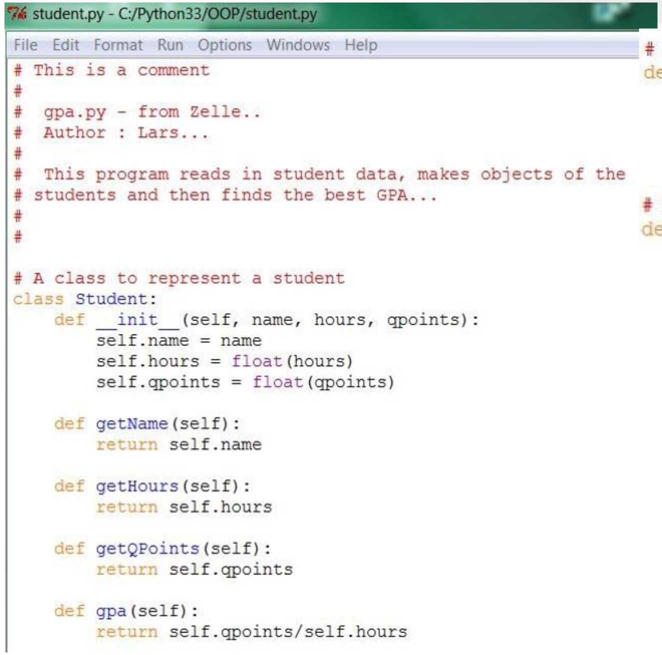
```
# This is a comment
#
#  gpa.py - from Zelle..
#  Author : Lars...
#
#  This program reads in student data, makes objects of the
# students and then finds the best GPA...
#
#

# A class to represent a student
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)

    def getName(self):
        return self.name

    def getHours(self):
        return self.hours

    def getQPoints(self):
        return self.qpoints

    def gpa(self):
        return self.qpoints/self.hours
```

```
# A function to take a student record and make an object of it
def makeStudent(infoStr):
    # infoStr is a tab separated line w name, hours, qpoints
    # return a student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

# Main Program
def main():
    # open the input file for reading
    filename = input("Enter the filename :")
    infile = open(filename, 'r')

    # set the best var to the first line
    best = makeStudent(infile.readline())

    # process the whole file
    for line in infile:
        s = makeStudent(line)
        if s.gpa() > best.gpa:
            best = s
    infile.close()

    # print the info to the screen
    print("The best student is ",best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())


# Why do we do this??
if __name__ == '__main__':
    main()
```

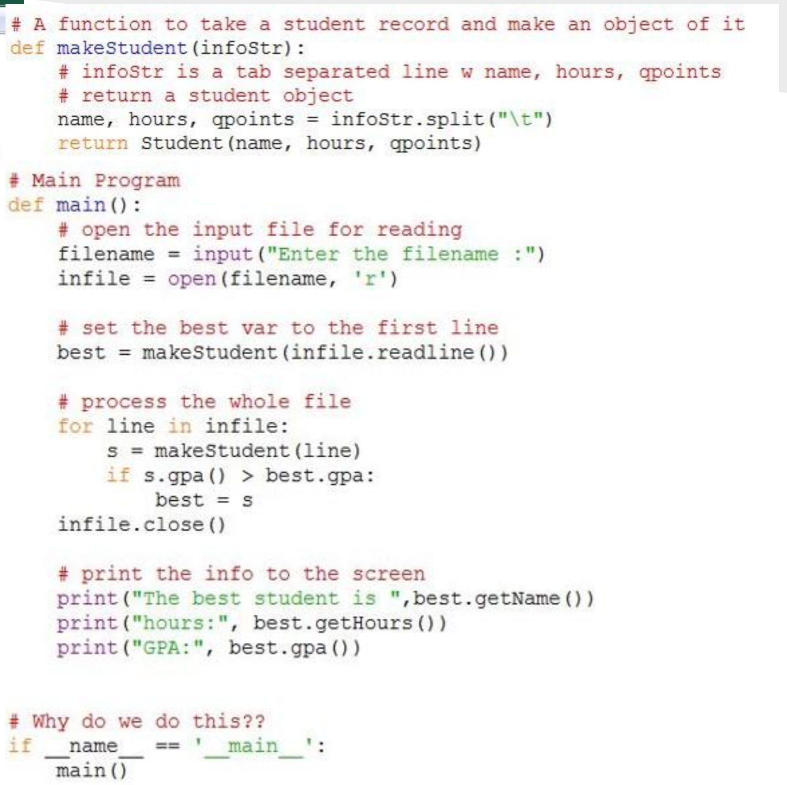
Python Methodologies for Data Sciences



```
Lars    9     25
Abhi    9     26
Evan    12    34
Julian  12    36
Saanvi  12    48
Mason   12    47
```


student.py - C:/Python33/OOP/student.py
File Edit Format Run Options Windows Help

```python
# This is a comment
#
#  gpa.py - from Zelle..
#  Author : Lars...
#
#  This program reads in student data, makes objects of the
# students and then finds the best GPA...
#
#


# A class to represent a student
class Student:
    def __init__(self, name, hours, qpoints):
        self.name = name
        self.hours = float(hours)
        self.qpoints = float(qpoints)

    def getName(self):
        return self.name

    def getHours(self):
        return self.hours

    def getQPoints(self):
        return self.qpoints

    def gpa(self):
        return self.qpoints/self.hours
```



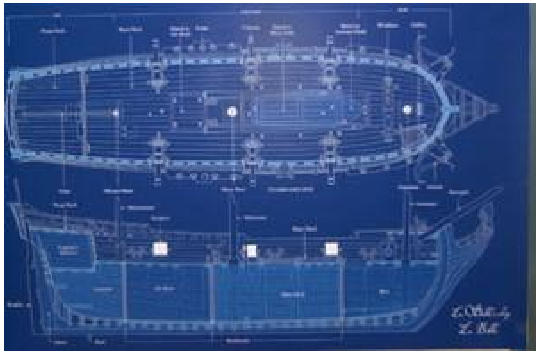
```python
# A function to take a student record and make an object of it
def makeStudent(infoStr):
    # infoStr is a tab separated line w name, hours, qpoints
    # return a student object
    name, hours, qpoints = infoStr.split("\t")
    return Student(name, hours, qpoints)

# Main Program
def main():
    # open the input file for reading
    filename = input("Enter the filename :")
    infile = open(filename, 'r')

    # set the best var to the first line
    best = makeStudent(infile.readline())

    # process the whole file
    for line in infile:
        s = makeStudent(line)
        if s.gpa() > best.gpa:
            best = s
    infile.close()

    # print the info to the screen
    print("The best student is ",best.getName())
    print("hours:", best.getHours())
    print("GPA:", best.gpa())


# Why do we do this??
if __name__ == '__main__':
    main()
```

Code this up and run it. Create a small file like the one I used up here to the left to run your program on.

Python Methodologies for Data Sciences

Object oriented programming is just representing objects as they would operate in the real world in our programs...

Remember, classes are just blueprints, blueprints that we use to create objects...



Our example was a die class, but we could have modelled anything!

Python Methodologies for Data Sciences

Make sure you absorb all of the proceeding before you move on. Go read Zelle. If I did the Unit 5 video go watch that. Make sure you "get it" before you go on to Inheritance and Polymorphism because a solid understanding of the basics of classes and objects is needed for these topics.

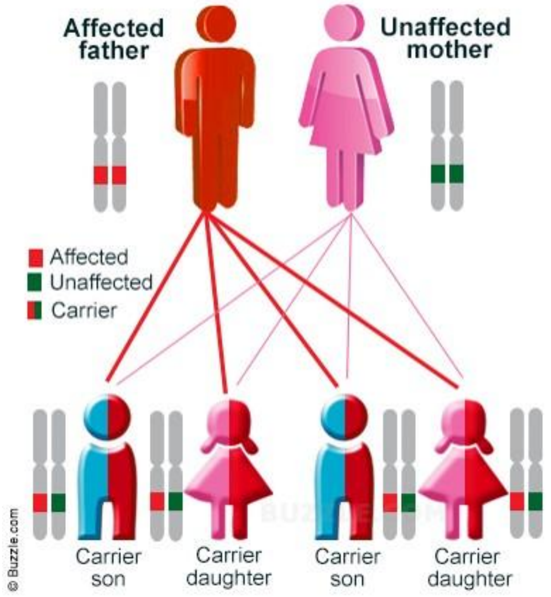
Enough scary professor crap, you'll be fine, just pay attention and don't let the funny words scare you...

Inheritance

Python Methodologies for Data Sciences

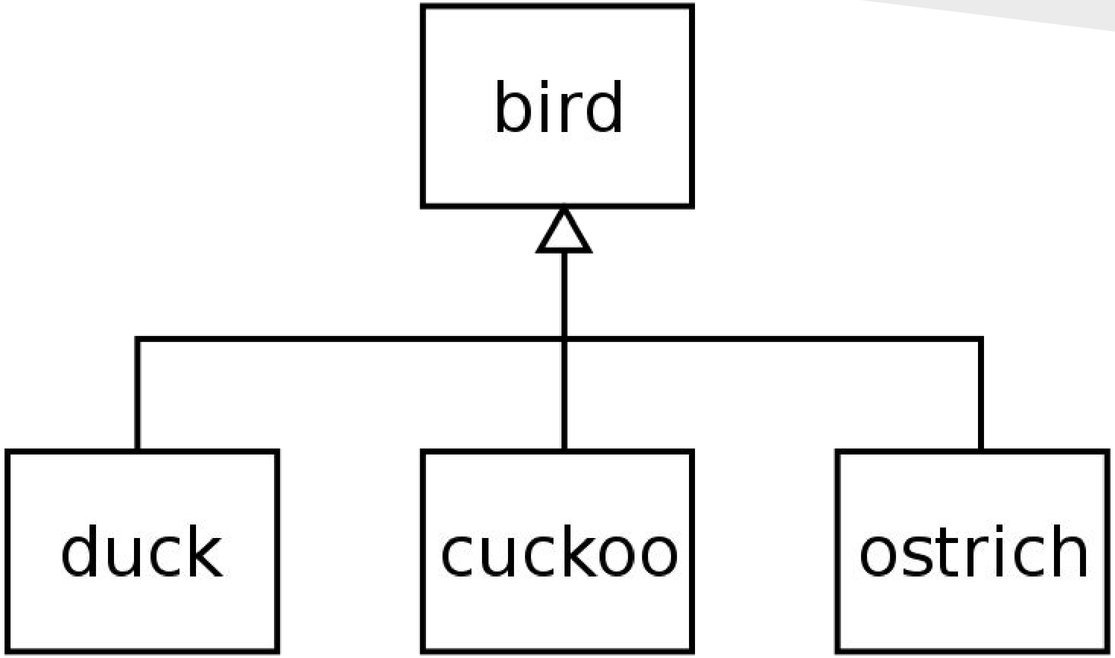
Python Methodologies for Data Sciences

Python Methodologies for Data Sciences

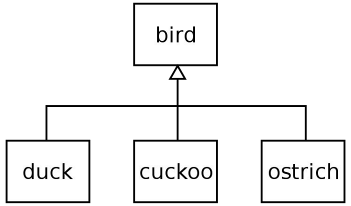


Python Methodologies for Data Sciences

Just what is inheritance?



Python Methodologies for Data Sciences

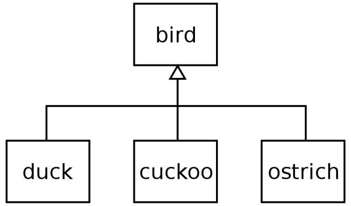


Inheritance is a way to save programmers a lot of work.

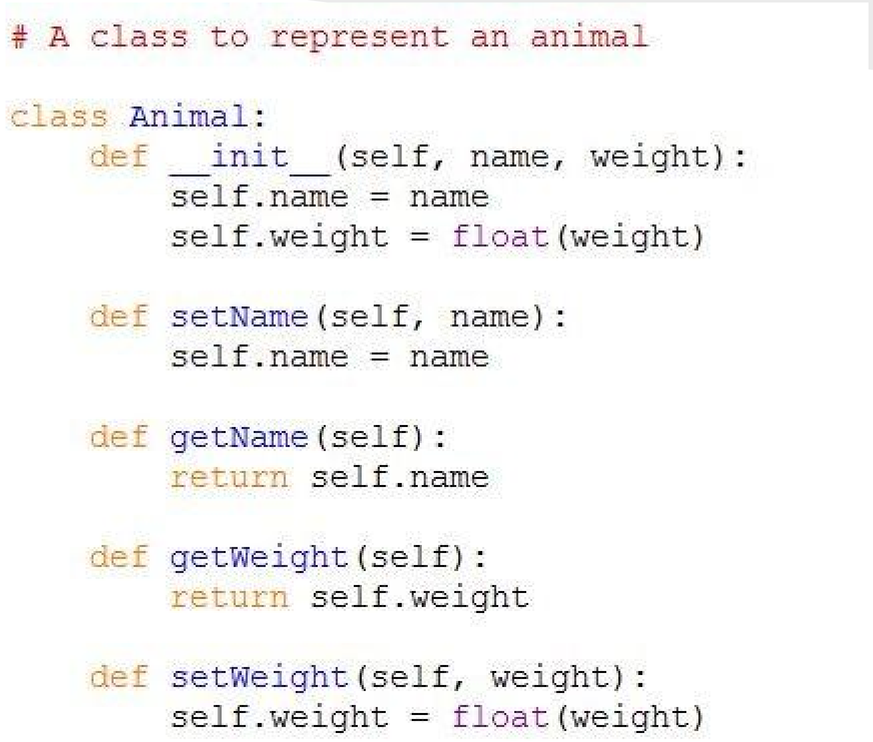
Inheritance allows us to extend classes that we have already written to make them more specific or to give them new functionality.

Object Orientation allows us to tell Python that we want to take a class and add things to it so we can have a more customized solution. But, we want to do this without changing the original class!

Python Methodologies for Data Sciences

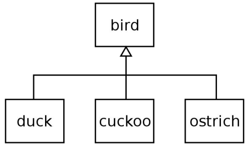


Take a class for an animal. We’ll keep it simple and just ask for a name and the animal’s weight.



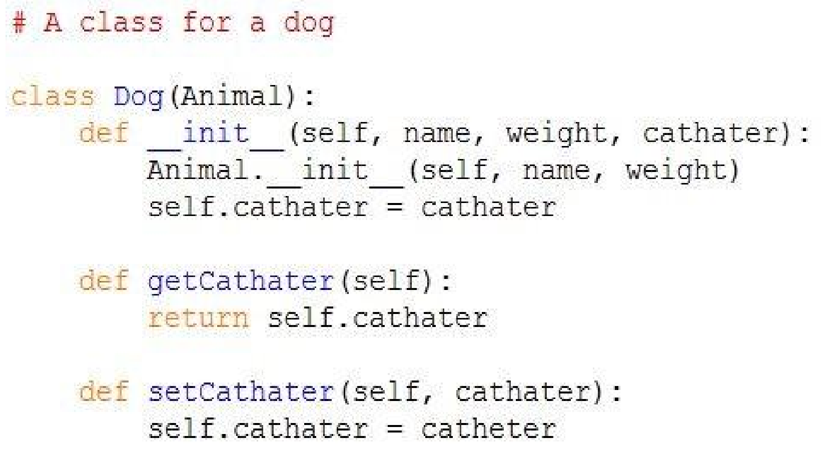
```
# A class to represent an animal

class Animal:
    def __init__(self, name, weight):
        self.name = name
        self.weight = float(weight)

    def setName(self, name):
        self.name = name

    def getName(self):
        return self.name

    def getWeight(self):
        return self.weight

    def setWeight(self, weight):
        self.weight = float(weight)
```

Python Methodologies for Data Sciences



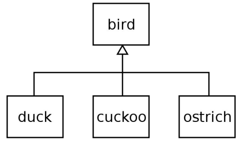
Now we want to create a class for a Dog. We don't want to retype all the class info and methods about name and weight, so we tell Python that we want to INHERIT from the Animal class.

Notice how we ask for all the information we'll need in the constructor. Then we call the constructor of the class we inherited from and give it the data it needs. THEN we set the variables for the Dog component.



```
# A class for a dog

class Dog(Animal):
    def __init__(self, name, weight, cathater):
        Animal.__init__(self, name, weight)
        self.cathater = cathater

    def getCathater(self):
        return self.cathater

    def setCathater(self, cathater):
        self.cathater = catheter
```

Python Methodologies for Data Sciences

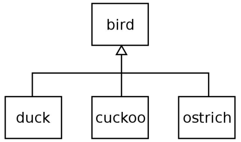


We call the Dog class a "subclass" and we call the Animal class a "superclass."
When we use an object we create with the subclass we can still use all of the methods and attributes from the superclass.
Let's take a look…

```
# A class for a dog

class Dog(Animal):
    def __init__(self, name, weight, cathater):
        Animal.__init__(self, name, weight)
        self.cathater = cathater

    def getCathater(self):
        return self.cathater

    def setCathater(self, cathater):
        self.cathater = catheter
```

```
# A class to represent an animal

class Animal:
    def __init__(self, name, weight):
        self.name = name
        self.weight = float(weight)

    def setName(self, name):
        self.name = name

    def getName(self):
        return self.name

    def getWeight(self):
        return self.weight

    def setWeight(self, weight):
        self.weight = float(weight)
```

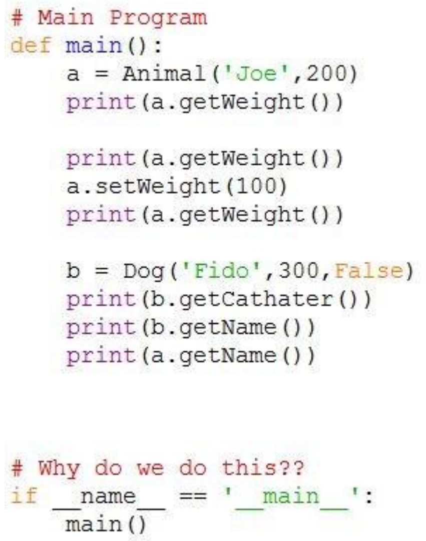
Python Methodologies for Data Sciences



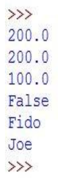
See how the first object is an “Animal.” We reset the weight and just print it.

The second object is a Dog object that inherited everything from Animal and added the Cat-Hater (not catheter like a tube, Cat-Hater as in this dog hates cats, long story, ask me sometime)

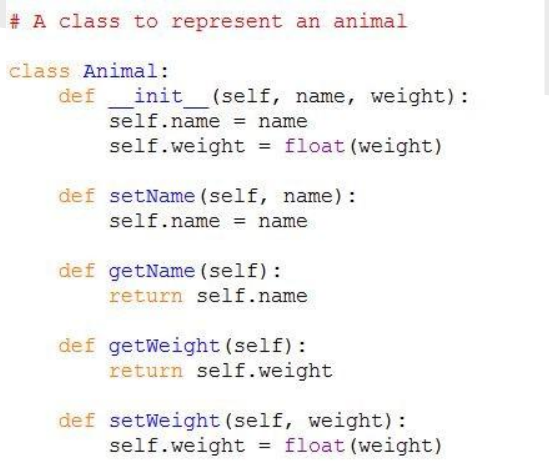
Fido is not a cat hater and as we can see from the second to last line, we are using the getName method we inherited from Animal in Dog.



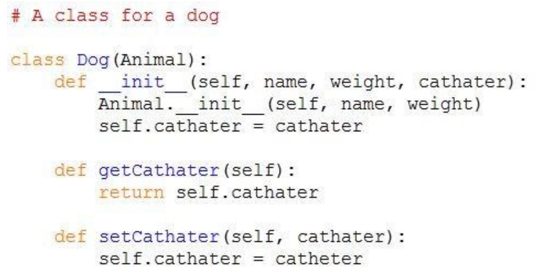
```
# Main Program
def main():
    a = Animal('Joe',200)
    print(a.getWeight())

    print(a.getWeight())
    a.setWeight(100)
    print(a.getWeight())

    b = Dog('Fido',300,False)
    print(b.getCathater())
    print(b.getName())
    print(a.getName())



# Why do we do this??
if __name__ == '__main__':
    main()
```



```
>>>
200.0
200.0
100.0
False
Fido
Joe
>>>
```

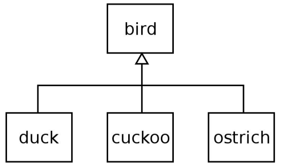


```
# A class to represent an animal

class Animal:
    def __init__(self, name, weight):
        self.name = name
        self.weight = float(weight)

    def setName(self, name):
        self.name = name

    def getName(self):
        return self.name

    def getWeight(self):
        return self.weight

    def setWeight(self, weight):
        self.weight = float(weight)
```



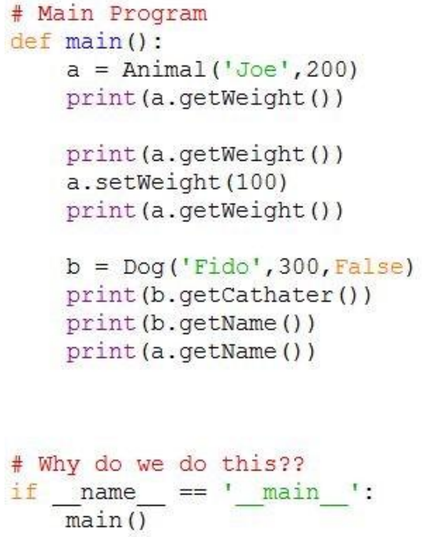
```
# A class for a dog

class Dog(Animal):
    def __init__(self, name, weight, cathater):
        Animal.__init__(self, name, weight)
        self.cathater = cathater

    def getCathater(self):
        return self.cathater

    def setCathater(self, cathater):
        self.cathater = catheter
```

Python Methodologies for Data Sciences

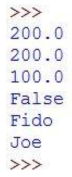


Inheritance is tricky so don’t be discouraged if you don’t get it right away.

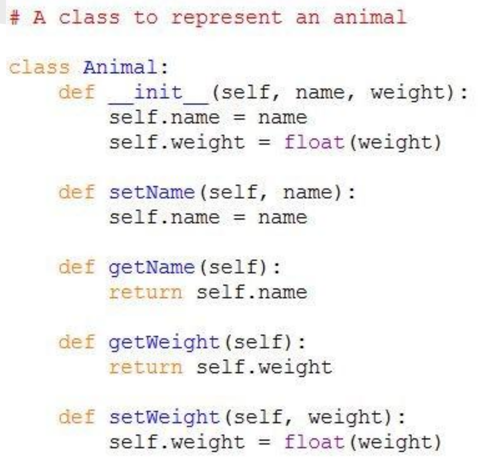
Code up the two classes here and play with them.



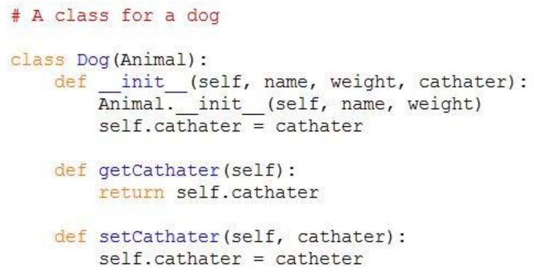
```
# Main Program
def main():
    a = Animal('Joe',200)
    print(a.getWeight())

    print(a.getWeight())
    a.setWeight(100)
    print(a.getWeight())

    b = Dog('Fido',300,False)
    print(b.getCathater())
    print(b.getName())
    print(a.getName())


# Why do we do this??
if __name__ == '__main__':
    main()
```



```
>>>
200.0
200.0
100.0
False
Fido
Joe
>>>
```

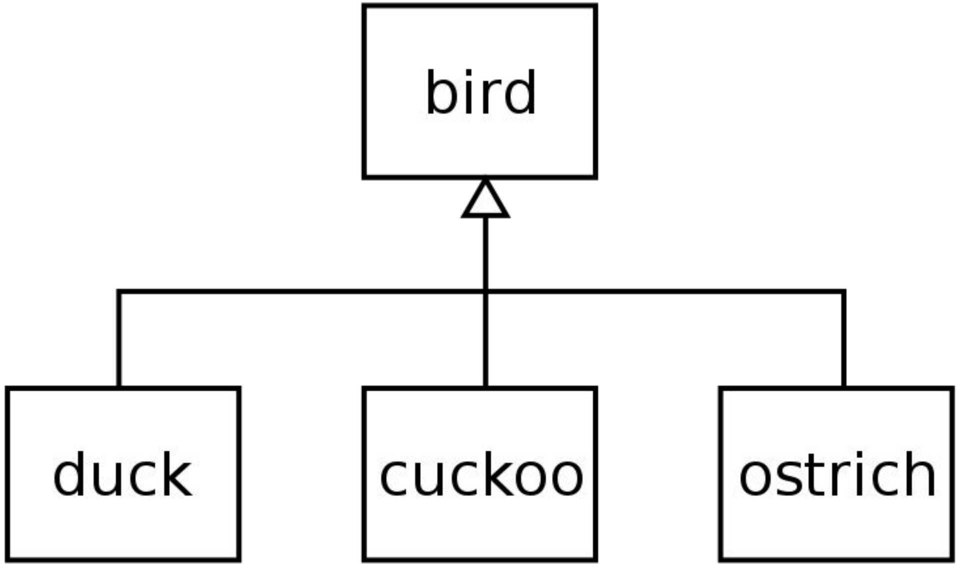


```
# A class to represent an animal

class Animal:
    def __init__(self, name, weight):
        self.name = name
        self.weight = float(weight)

    def setName(self, name):
        self.name = name

    def getName(self):
        return self.name

    def getWeight(self):
        return self.weight

    def setWeight(self, weight):
        self.weight = float(weight)
```



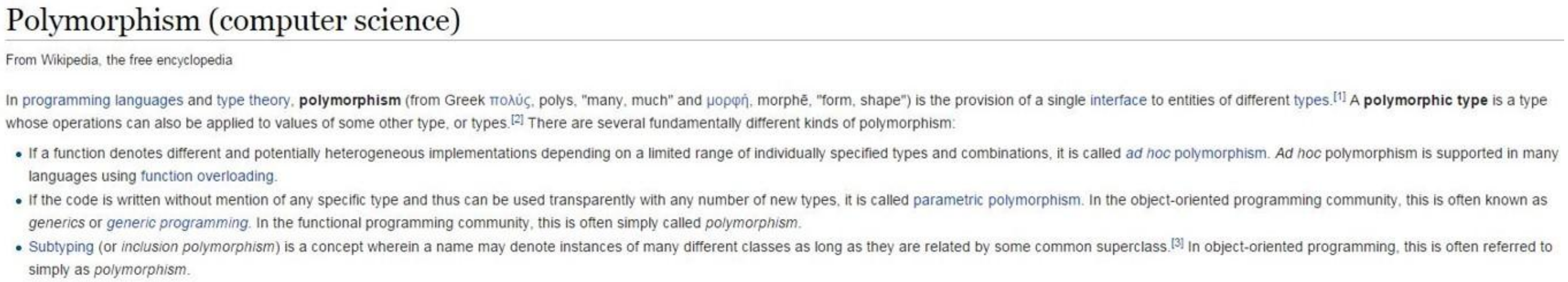
```
# A class for a dog

class Dog(Animal):
    def __init__(self, name, weight, cathater):
        Animal.__init__(self, name, weight)
        self.cathater = cathater

    def getCathater(self):
        return self.cathater

    def setCathater(self, cathater):
        self.cathater = catheter
```

Python Methodologies for Data Sciences



Polymorphism

What is Polymorphism?


Polymorphism (computer science)

From Wikipedia, the free encyclopedia

In programming languages and type theory, **polymorphism** (from Greek πολύς, polys, "many, much" and μορφή, morphē, "form, shape") is the provision of a single interface to entities of different types.[1] A **polymorphic type** is a type whose operations can also be applied to values of some other type, or types.[2] There are several fundamentally different kinds of polymorphism:

- If a function denotes different and potentially heterogeneous implementations depending on a limited range of individually specified types and combinations, it is called *ad hoc* polymorphism. *Ad hoc* polymorphism is supported in many languages using function overloading.
- If the code is written without mention of any specific type and thus can be used transparently with any number of new types, it is called parametric polymorphism. In the object-oriented programming community, this is often known as *generics* or *generic programming*. In the functional programming community, this is often simply called *polymorphism*.
- Subtyping (or *inclusion polymorphism*) is a concept wherein a name may denote instances of many different classes as long as they are related by some common superclass.[3] In object-oriented programming, this is often referred to simply as *polymorphism*.

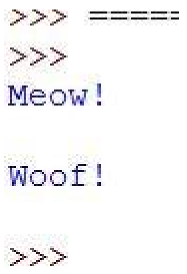
Wow, that’s pretty confusing ivory tower CS nerd talk right there.
Let’s just look at some code (as usual...)

Python Methodologies for Data Sciences

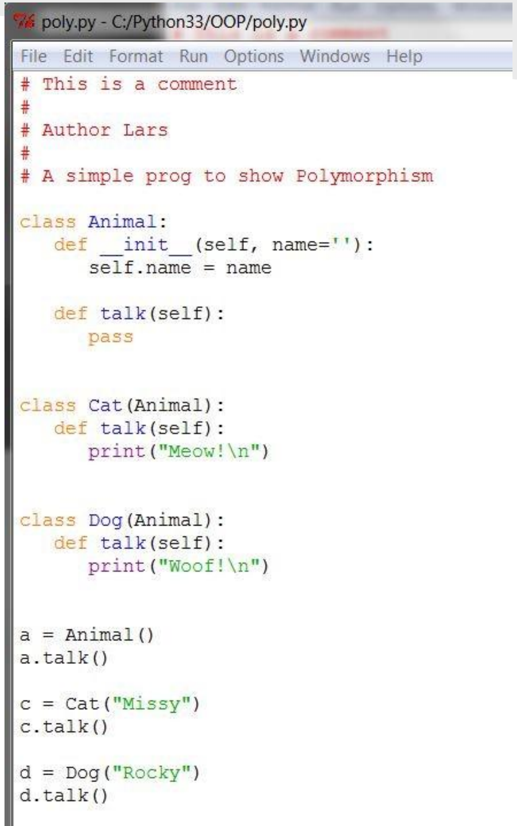
Polymorphism means “many forms”and that’s pretty much the case. I know that Animal has a method called “talk” but it’s implemented differently by the sub-classes depending on that class’s particular behaviors. All animals “talk” (those that don’t could print ‘’ I guess) but in the individual sub class we determine what they sound like.

If I knew my object was of type Animal but was not sure if it were a dog or cat I would just run talk() and find out. This kind of method would be used for a character in a video game.

Having a method with the same name do different things depending on the object’s type is a case of Polymorphism!



```
>>> =====
>>>
Meow!

Woof!

>>>
```


poly.py - C:/Python33/OOP/poly.py
File Edit Format Run Options Windows Help

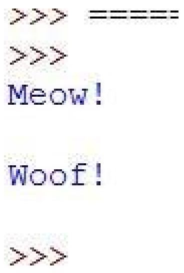
```
# This is a comment
#
# Author Lars
#
# A simple prog to show Polymorphism

class Animal:
    def __init__(self, name=''):
        self.name = name

    def talk(self):
        pass


class Cat(Animal):
    def talk(self):
        print("Meow!\n")


class Dog(Animal):
    def talk(self):
        print("Woof!\n")


a = Animal()
a.talk()

c = Cat("Missy")
c.talk()

d = Dog("Rocky")
d.talk()
```

Python Methodologies for Data Sciences

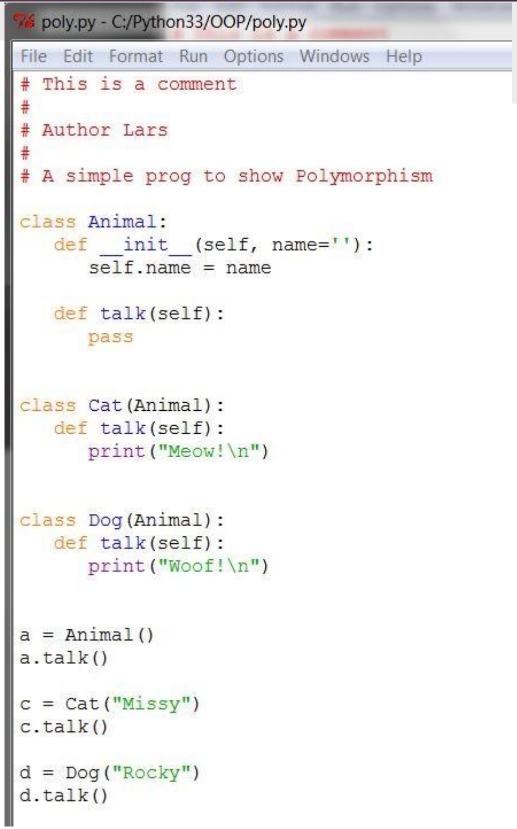
The subclass is "overriding" the method in the superclass. It's like the Cat class saying: "I have my own talk method, thank you very much. If someone instantiates a Cat object, we'll use my talk method if it's called on that variable name."

It's the same with Dog.

But, in the absence of a talk method in the sub class we would revert to the Super class's behavior, in this case doing nothing (see the pass?)



```
>>> =====
>>>
Meow!

Woof!

>>>
```



poly.py - C:/Python33/OOP/poly.py
File Edit Format Run Options Windows Help

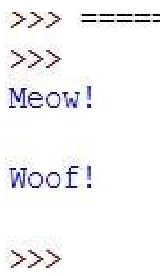
```
# This is a comment
#
# Author Lars
#
# A simple prog to show Polymorphism

class Animal:
    def __init__(self, name=''):
        self.name = name

    def talk(self):
        pass


class Cat(Animal):
    def talk(self):
        print("Meow!\n")


class Dog(Animal):
    def talk(self):
        print("Woof!\n")


a = Animal()
a.talk()

c = Cat("Missy")
c.talk()

d = Dog("Rocky")
d.talk()
```

Python Methodologies for Data Sciences

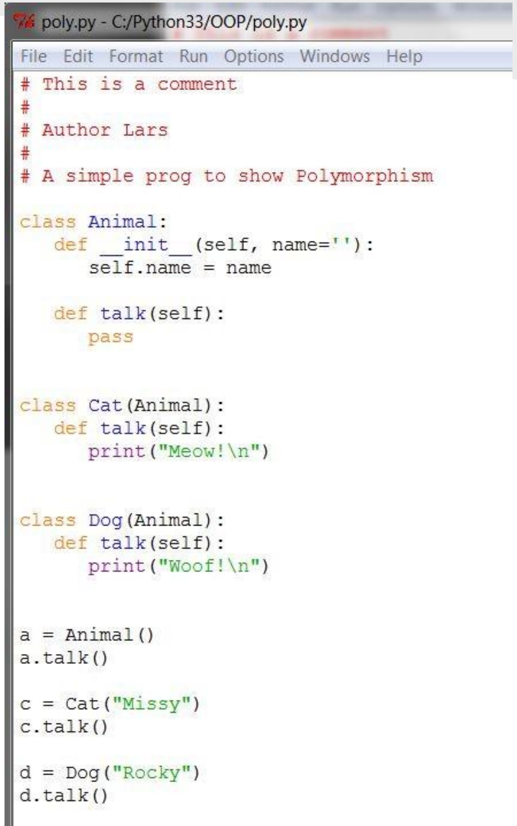
Two things we want you to see here.

1. Look at the talk method in Animal. It is undefined, we just use a pass. This is considered Python's way of implementing abstract classes (My Java and C++ people in the house!). No object was ever meant to be instantiated with Animal, the programmer uses the class as a base for subclasses to inherit from, that's all.

2. We never call Animal's constructor with the animal names when we use Cat and Dog. How are these objects getting these values? It's because we ALWAYS have a default constructor that we inherit from (the superclass) if we do not create one of our own.



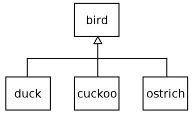
```
>>> =====
>>>
Meow!

Woof!

>>>
```


poly.py - C:/Python33/OOP/poly.py
File Edit Format Run Options Windows Help

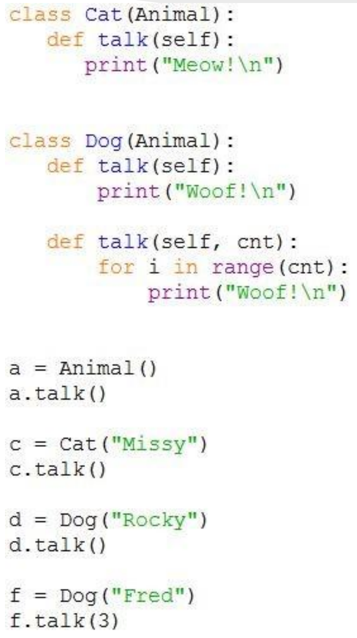
```python
# This is a comment
#
# Author Lars
#
# A simple prog to show Polymorphism

class Animal:
    def __init__(self, name=''):
        self.name = name

    def talk(self):
        pass


class Cat(Animal):
    def talk(self):
        print("Meow!\n")


class Dog(Animal):
    def talk(self):
        print("Woof!\n")


a = Animal()
a.talk()

c = Cat("Missy")
c.talk()

d = Dog("Rocky")
d.talk()
```

Python Methodologies for Data Sciences

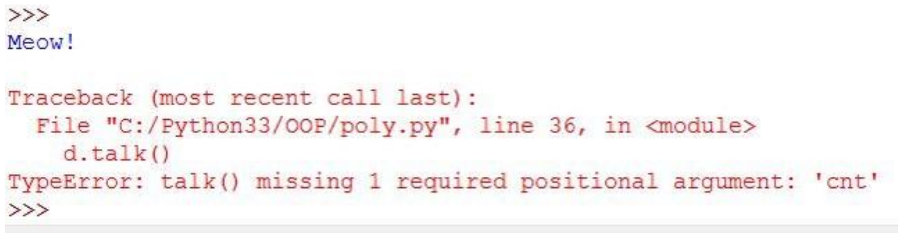
No Overloading



Other computer languages (Java, C++) have support for something called method overloading. That's when two methods with the same name but differing parameter lists can be separately defined and have unique behaviors. Python does NOT support this kind of polymorphism, there is no function overloading in Python.



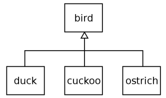
```
class Cat(Animal):
   def talk(self):
      print("Meow!\n")


class Dog(Animal):
   def talk(self):
       print("Woof!\n")

   def talk(self, cnt):
       for i in range(cnt):
           print("Woof!\n")


a = Animal()
a.talk()

c = Cat("Missy")
c.talk()

d = Dog("Rocky")
d.talk()

f = Dog("Fred")
f.talk(3)
```



```
>>>
Meow!

Traceback (most recent call last):
  File "C:/Python33/OOP/poly.py", line 36, in <module>
    d.talk()
TypeError: talk() missing 1 required positional argument: 'cnt'
>>>
```

Python Methodologies for Data Sciences

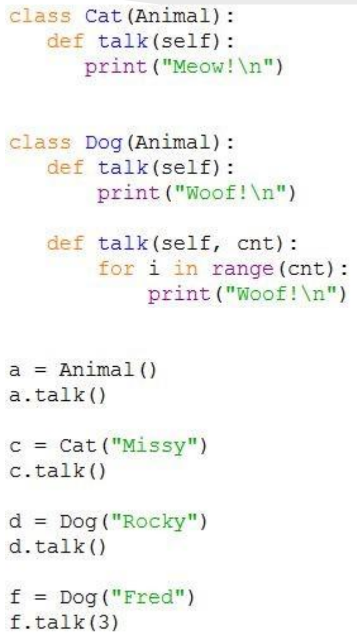
No Overloading



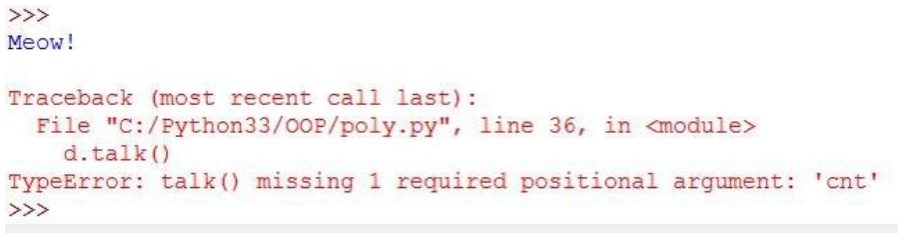
In Python you cannot overload a method within a class by giving it a unique set of parameters.

BUT

You can override methods from classes that you have inherited from, just like we do with talk in Cat and Dog. We could have used any parameter list we wanted.

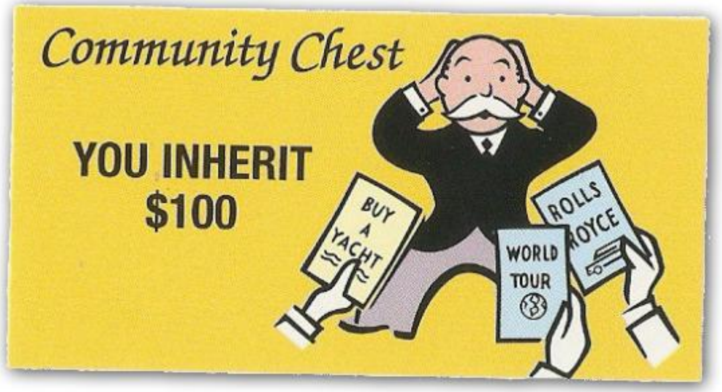


```
class Cat(Animal):
    def talk(self):
        print("Meow!\n")


class Dog(Animal):
    def talk(self):
        print("Woof!\n")

    def talk(self, cnt):
        for i in range(cnt):
            print("Woof!\n")


a = Animal()
a.talk()

c = Cat("Missy")
c.talk()

d = Dog("Rocky")
d.talk()

f = Dog("Fred")
f.talk(3)
```

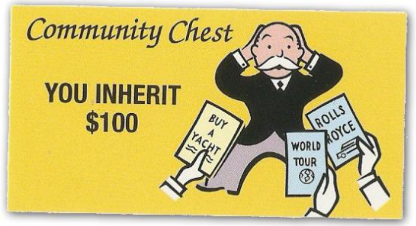


```
>>>
Meow!

Traceback (most recent call last):
  File "C:/Python33/OOP/poly.py", line 36, in <module>
    d.talk()
TypeError: talk() missing 1 required positional argument: 'cnt'
>>>
```

Python Methodologies for Data Sciences



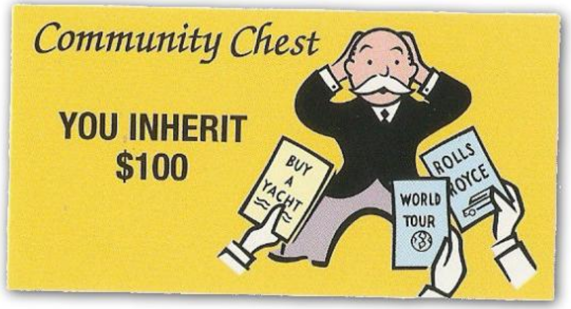
Python Methodologies for Data Sciences



Inheritance is when you use another class as a 'base' and merely add on to it's functionality by 'extend'ing that class. The class being inherited from is the "superclass" and the class doing the inheriting is the "subclass." Abstract classes and methods are made merely to be inherited from and not to actually have objects instantiated from them.

Polymorphism is when methods with the same name perform differently based on the object type. In our example we defined the talk() method in an animal class as abstract. In our sub classes, Cat and Dog we overrode that abstract method and created two new methods, both with the same name, but both behaving as their type would dictate.

Python Methodologies for Data Sciences



Take your time with Inheritance and Polymorphism.

Python Methodologies for Data Sciences

As a matter of fact, take your time by taking a break all together. Now you know what OOP is. You know what classes are. Now you know what Inheritance and Polymorphism are.

All we have to do now is clean up a bit. Take a break and I'll show you some goodies. Things like a variable that all objects from the same class can share, a method that runs when an object is destroyed (yes, we can destroy objects, it's one of life's true pleasures) and we are going to talk more about the devious practice of overriding the superclasses functions to make them do what we want.

Some Extra Goodies

Python Methodologies for Data Sciences

Class Variables

Deconstructors

More Overriding Functions

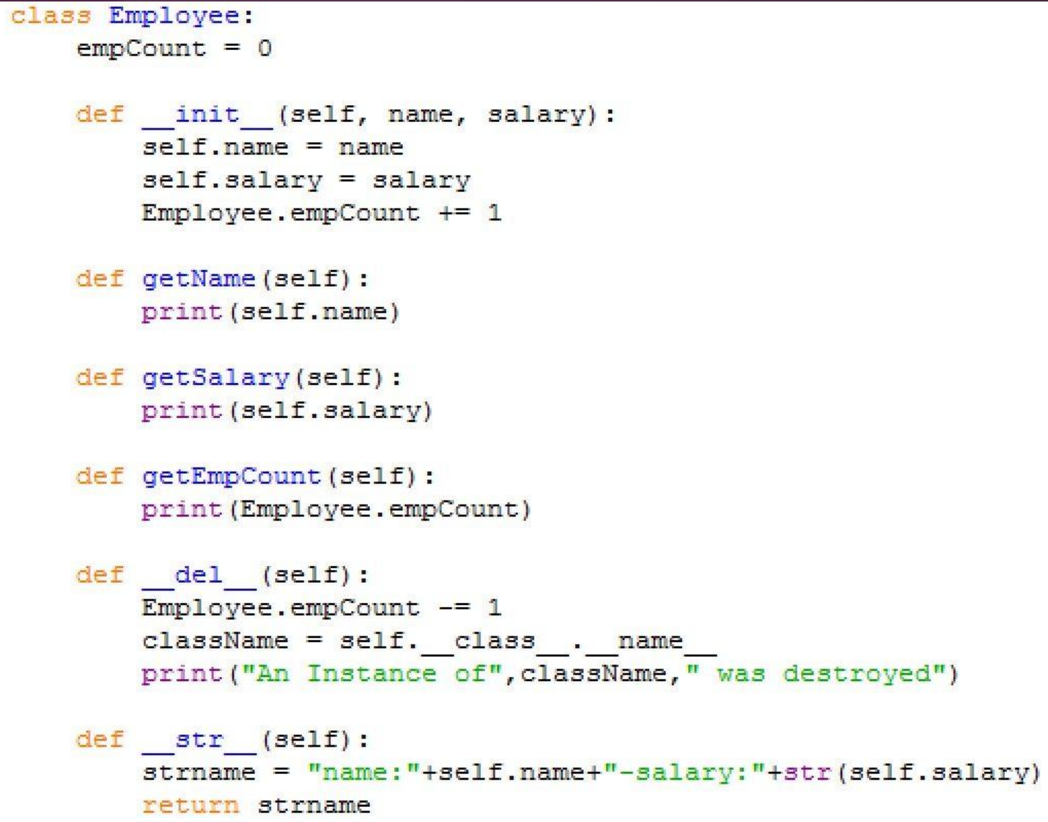
Python Methodologies for Data Sciences

We could take all night with this and do an individual program for each of these features...

Python Methodologies for Data Sciences

Class Variables

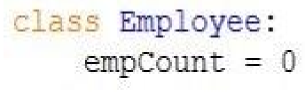
Or we could create a Super Program that has it all!!



```python
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

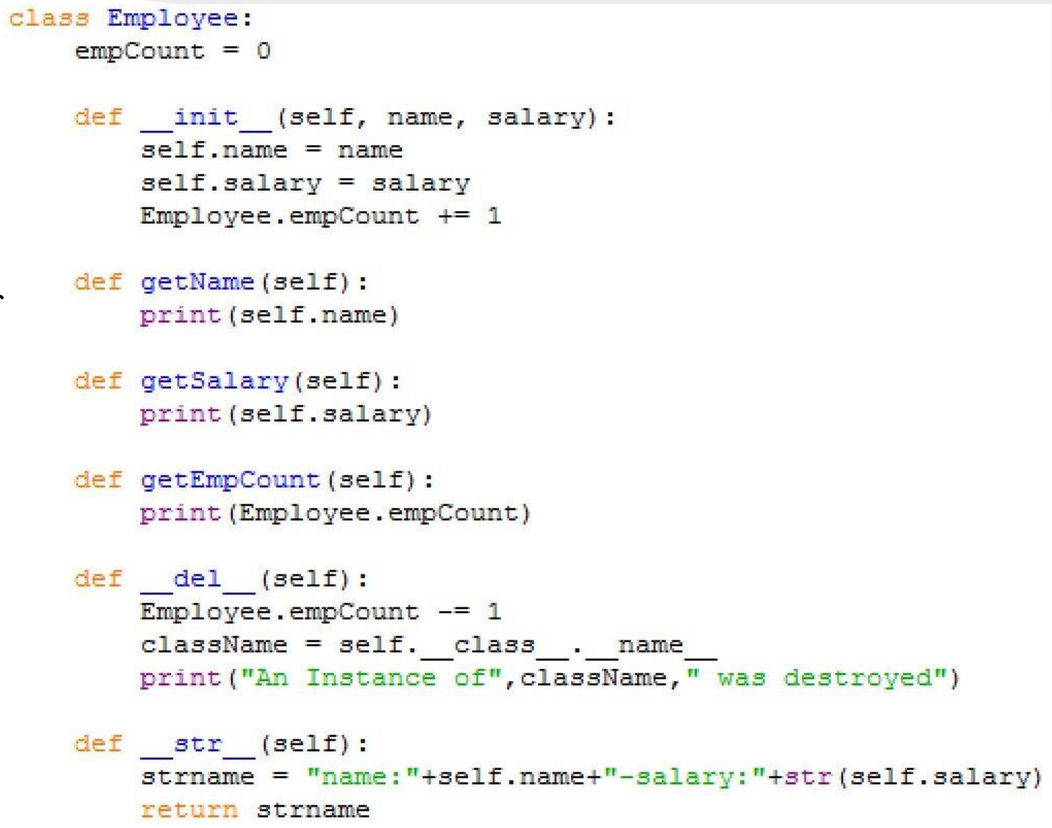
Class Variables



```
class Employee:
    empCount = 0
```

A class variable, shown above, is created outside of the constructor method and can be accessed by any object of the class.

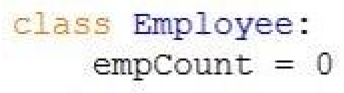
Note that in this case we use the variable to keep track of how many objects we have, so the constructor increments the variable by one. “+=” means “add what’s on the right to the variable on the left of the symbol.



```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

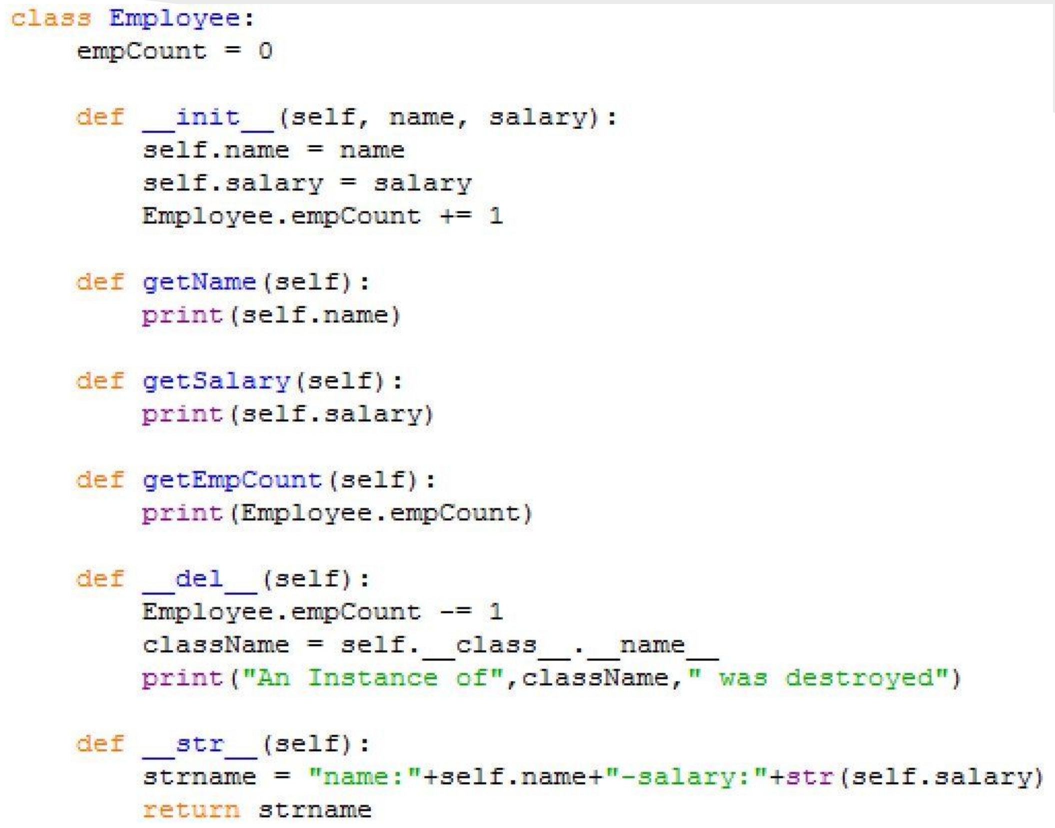
Class Variables



```
class Employee:
    empCount = 0
```

Notice the method "getEmpCount()" We print the class variable not by saying self.empCount, but by using the class name, Employee.empCount.

While we do not do so here, using class variables is a good way to send messages to other objects. You could have a boolean variable that tells all the other objects whether something is true or false. Class variables are often used in video games to track the number of objects that have been instantiated, be it number of enemies, shared money, etc.



```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

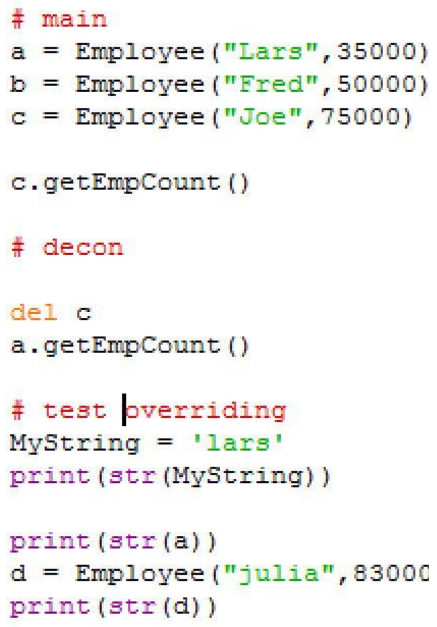
Python Methodologies for Data Sciences

Deconstructors

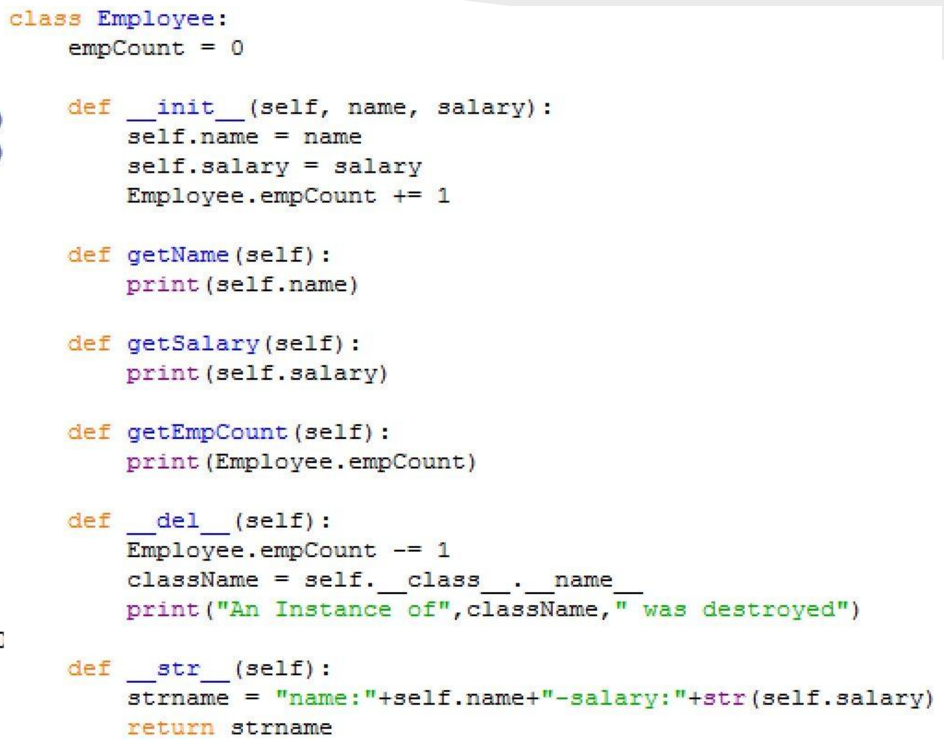
Look in the middle of the code here. See the ‘del’ command. It’s a keyword you’ll notice.

It’s short for ‘delete’ and it is the command we use when we want to “destroy” an object, get rid of it completely and give its memory back to the program.

We can tell the object what to do when it’s deleted by using a “deconstructor.”



```
# main
a = Employee("Lars",35000)
b = Employee("Fred",50000)
c = Employee("Joe",75000)

c.getEmpCount()

# decon

del c
a.getEmpCount()

# test overriding
MyString = 'lars'
print(str(MyString))

print(str(a))
d = Employee("julia",8300
print(str(d))
```



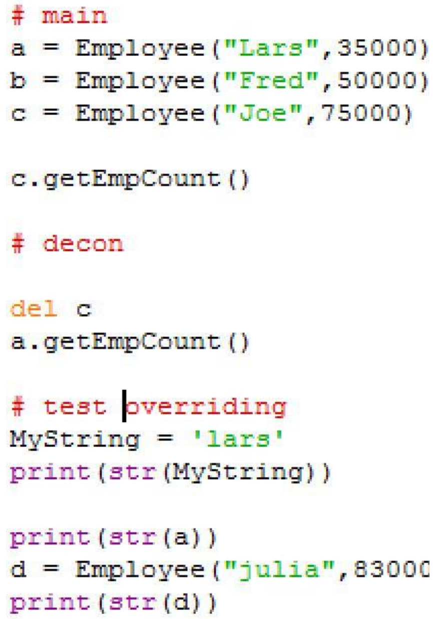
```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

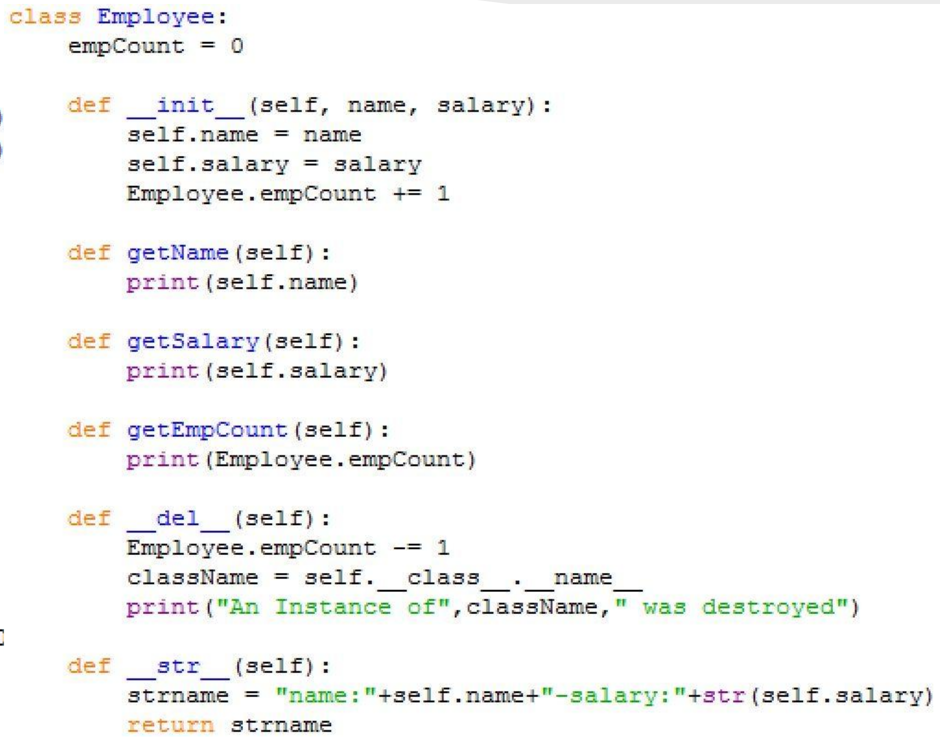
Deconstructors

Look in the class. See the method __del__? That's a deconstructor. Inside we decrement the class variable and print that we have destroyed the object.

What we do inside of __del__ is what happens when we delete the object.



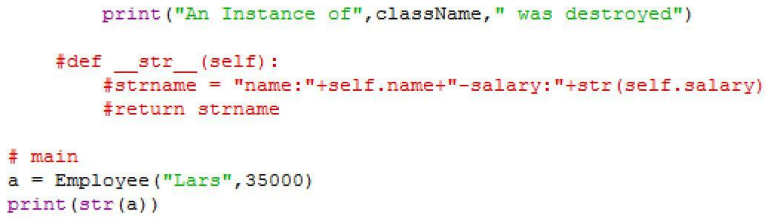
```
# main
a = Employee("Lars",35000)
b = Employee("Fred",50000)
c = Employee("Joe",75000)

c.getEmpCount()

# decon

del c
a.getEmpCount()

# test overriding
MyString = 'lars'
print(str(MyString))

print(str(a))
d = Employee("julia",83000
print(str(d))
```



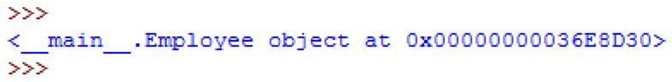
```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

Overriding Functions



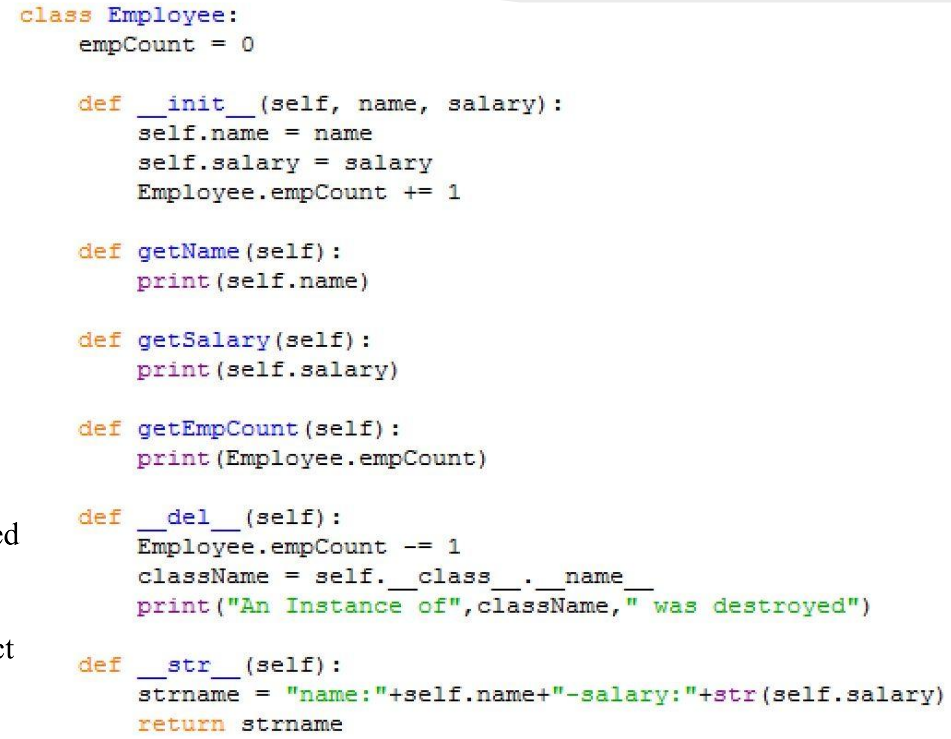
```
            print("An Instance of",className," was destroyed")

        #def __str__(self):
            #strname = "name:"+self.name+"-salary:"+str(self.salary)
            #return strname

    # main
    a = Employee("Lars",35000)
    print(str(a))
```



```
>>>
<__main__.Employee object at 0x00000000036E8D30>
>>>
```

Here I comment out the __str__ method to show you what gets printed when we try to “print” the string version of an object.

It looks confusing, but it’s just the memory location where the object resides. __str__ is a method all classes have. It tells Python what to print when the “print” function is run on the object.

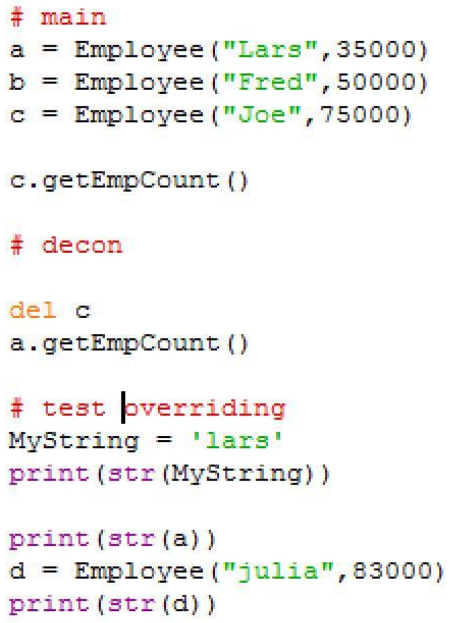


```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

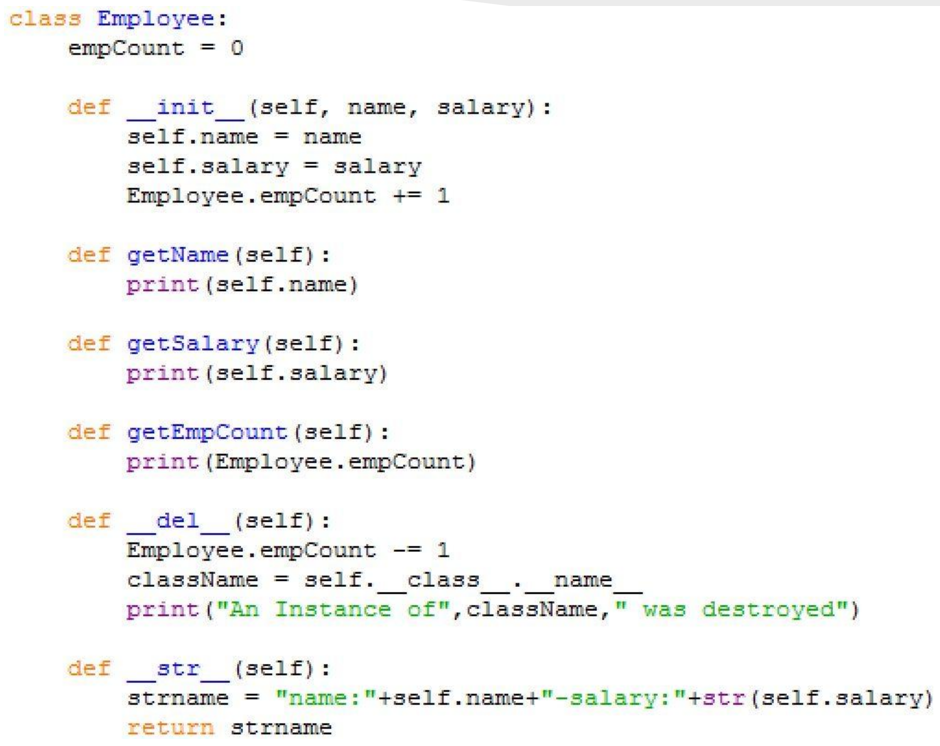
Python Methodologies for Data Sciences

Overriding Functions

When we print our objects we just want to show what the data item values are. We do this by ‘overriding’ the __str__ method and writing our own in this class.



```
# main
a = Employee("Lars",35000)
b = Employee("Fred",50000)
c = Employee("Joe",75000)

c.getEmpCount()

# decon

del c
a.getEmpCount()

# test overriding
MyString = 'lars'
print(str(MyString))

print(str(a))
d = Employee("julia",83000)
print(str(d))
```



```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

Overriding Functions

Look below to see the output of the code. Code it up and run it yourself.

See how the empCount decrements after the ‘del’ command? See how we now get data values when we send our object variables to the print function?

If all of this confuses you do not fear. I will be going through this code in a review video.

```
# main
a = Employee("Lars",35000)
b = Employee("Fred",50000)
c = Employee("Joe",75000)

c.getEmpCount()

# decon

del c
a.getEmpCount()

# test overriding
MyString = 'lars'
print(str(MyString))

print(str(a))
d = Employee("julia",83000)
print(str(d))
```

```
3
An Instance of Employee  was destroyed
2
lars
name:LarsSalary35000
name:juliaSalary83000
>>>
```

```
class Employee:
    empCount = 0

    def __init__(self, name, salary):
        self.name = name
        self.salary = salary
        Employee.empCount += 1

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

    def getSalary(self):
        print(self.salary)

    def getEmpCount(self):
        print(Employee.empCount)

    def __del__(self):
        Employee.empCount -= 1
        className = self.__class__.__name__
        print("An Instance of",className," was destroyed")

    def __str__(self):
        strname = "name:"+self.name+"-salary:"+str(self.salary)
        return strname
```

Python Methodologies for Data Sciences

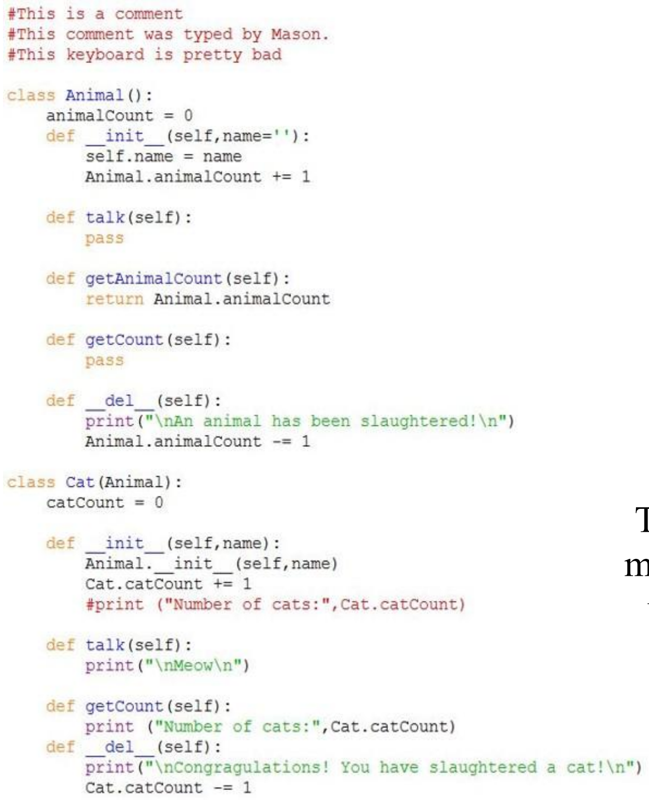
Class Variables - variables that can be accessed across objects that came from that class. Class variables are often altered in constructors and deconstructors and can also be used to send messages between objects.

Deconstructors - The opposite of constructors. This method is run when an object is deleted.

Overriding functions - When an inherited function does not have the desired behavior we can write our own function with the same name, telling Python that “If I instantiate an object with this class and you run this method, use mine and not the inherited one.”

The Big Finale

Python Methodologies for Data Sciences

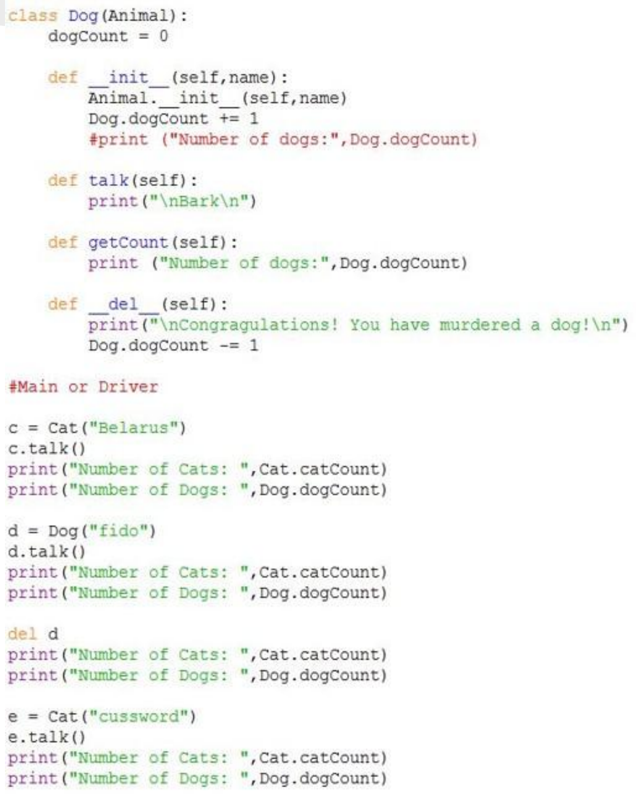


```python
#This is a comment
#This comment was typed by Mason.
#This keyboard is pretty bad

class Animal():
    animalCount = 0
    def __init__(self,name=''):
        self.name = name
        Animal.animalCount += 1

    def talk(self):
        pass

    def getAnimalCount(self):
        return Animal.animalCount

    def getCount(self):
        pass

    def __del__(self):
        print("\nAn animal has been slaughtered!\n")
        Animal.animalCount -= 1

class Cat(Animal):
    catCount = 0

    def __init__(self,name):
        Animal.__init__(self,name)
        Cat.catCount += 1
        #print ("Number of cats:",Cat.catCount)

    def talk(self):
        print("\nMeow\n")

    def getCount(self):
        print ("Number of cats:",Cat.catCount)
    def __del__(self):
        print("\nCongragulations! You have slaughtered a cat!\n")
        Cat.catCount -= 1
```

This is everything.

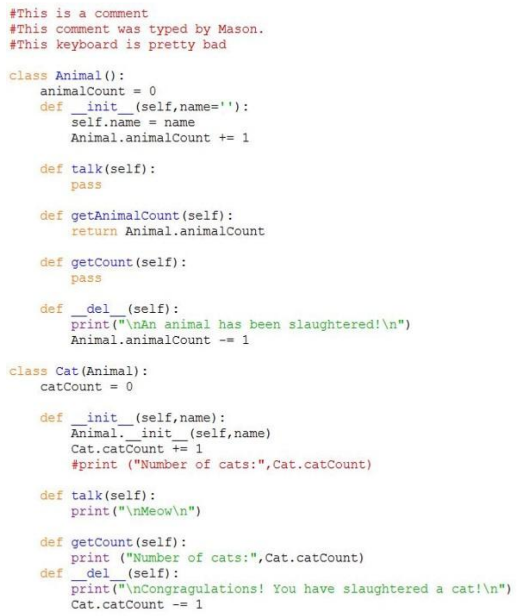
Classes, instance data, methods, class variables, inheritance, polymorphism, deconstructors, overriding of functions, this program has everything we’ve discussed.

The classes are defined and the main part of the program merely uses them by creating objects and using their methods...



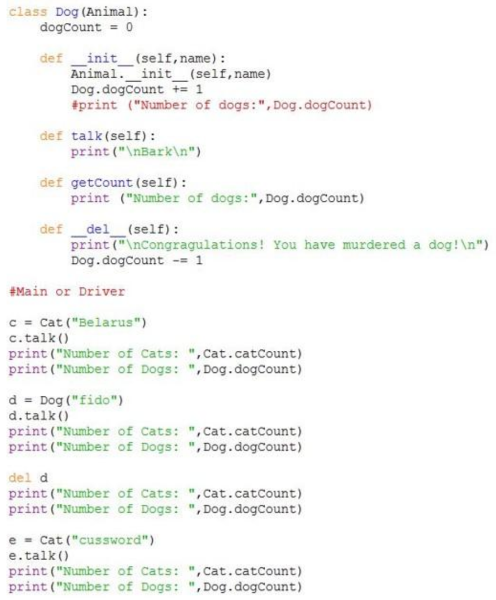
```python
class Dog(Animal):
    dogCount = 0

    def __init__(self,name):
        Animal.__init__(self,name)
        Dog.dogCount += 1
        #print ("Number of dogs:",Dog.dogCount)

    def talk(self):
        print("\nBark\n")

    def getCount(self):
        print ("Number of dogs:",Dog.dogCount)

    def __del__(self):
        print("\nCongragulations! You have murdered a dog!\n")
        Dog.dogCount -= 1

#Main or Driver

c = Cat("Belarus")
c.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

d = Dog("fido")
d.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

del d
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

e = Cat("cussword")
e.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)
```

Python Methodologies for Data Sciences



```python
#This is a comment
#This comment was typed by Mason.
#This keyboard is pretty bad

class Animal():
    animalCount = 0
    def __init__(self,name=''):
        self.name = name
        Animal.animalCount += 1

    def talk(self):
        pass

    def getAnimalCount(self):
        return Animal.animalCount

    def getCount(self):
        pass

    def __del__(self):
        print("\nAn animal has been slaughtered!\n")
        Animal.animalCount -= 1

class Cat(Animal):
    catCount = 0

    def __init__(self,name):
        Animal.__init__(self,name)
        Cat.catCount += 1
        #print ("Number of cats:",Cat.catCount)

    def talk(self):
        print("\nMeow\n")

    def getCount(self):
        print ("Number of cats:",Cat.catCount)
    def __del__(self):
        print("\nCongragulations! You have slaughtered a cat!\n")
        Cat.catCount -= 1
```

How can we make this better?

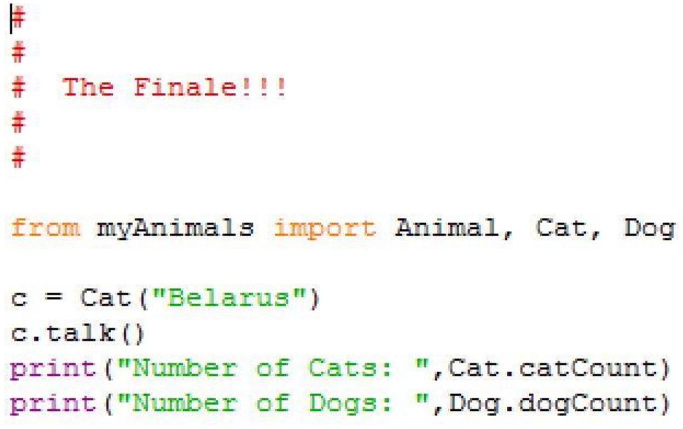


```python
class Dog(Animal):
    dogCount = 0

    def __init__(self,name):
        Animal.__init__(self,name)
        Dog.dogCount += 1
        #print ("Number of dogs:",Dog.dogCount)

    def talk(self):
        print("\nBark\n")

    def getCount(self):
        print ("Number of dogs:",Dog.dogCount)

    def __del__(self):
        print("\nCongragulations! You have murdered a dog!\n")
        Dog.dogCount -= 1

#Main or Driver

c = Cat("Belarus")
c.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

d = Dog("fido")
d.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

del d
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)

e = Cat("cussword")
e.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)
```

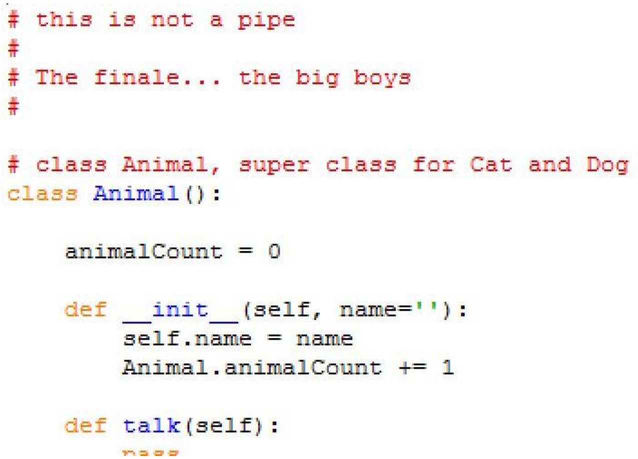
Modularity!!

Create a file called myAnimals.py. Put the Animal, Cat and Dog classes in it. Save it!

Create a program called finale.py and import your classes. Look at the first line! The rules are EXACTLY the same as when you import functions or constant variables. Now you can import CLASSES!



```
#
#
#  The Finale!!!
#
#

from myAnimals import Animal, Cat, Dog

c = Cat("Belarus")
c.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)
```



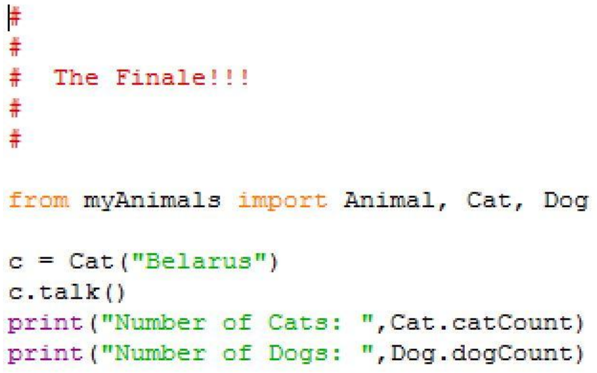
```
# this is not a pipe
#
# The finale... the big boys
#

# class Animal, super class for Cat and Dog
class Animal():

    animalCount = 0

    def __init__(self, name=''):
        self.name = name
        Animal.animalCount += 1

    def talk(self):
        pass
```

Python Methodologies for Data Sciences

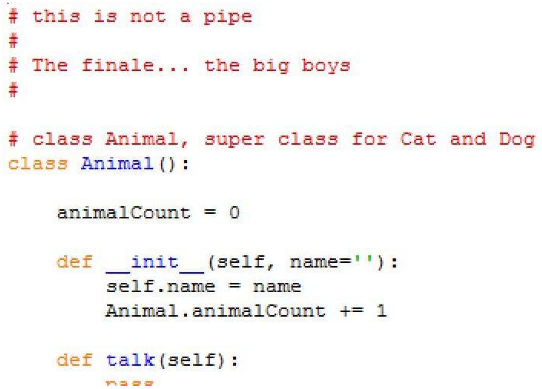
Modularity!!

I’ll meet you halfway. I put my copy of myAnimals.py in the resource folder for Unit5.

Now you code up finale.py and try it out.
(ps - that’s not all the code down there, go back a few slides for all of it)



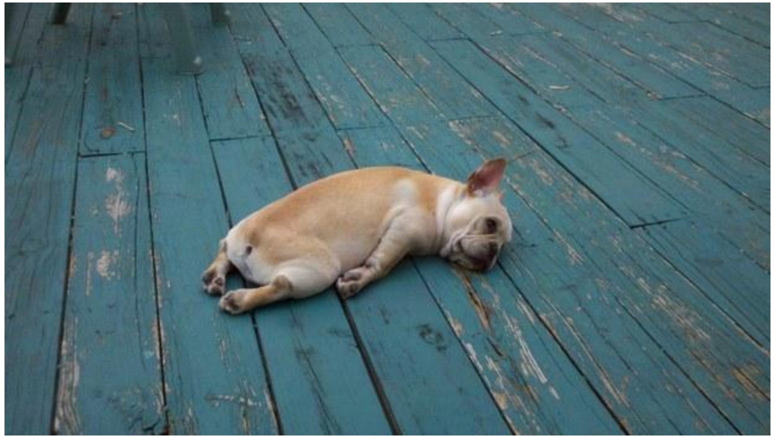
```
#
#
#   The Finale!!!
#
#

from myAnimals import Animal, Cat, Dog

c = Cat("Belarus")
c.talk()
print("Number of Cats: ",Cat.catCount)
print("Number of Dogs: ",Dog.dogCount)
```



```
# this is not a pipe
#
# The finale... the big boys
#

# class Animal, super class for Cat and Dog
class Animal():

    animalCount = 0

    def __init__(self, name=''):
        self.name = name
        Animal.animalCount += 1

    def talk(self):
```

Python Methodologies for Data Sciences

You just covered the basics of object oriented programming in one unit. If you're not exhausted there's something wrong with you.



To sum things up

This unit was big with an ambitious goal. To teach you OOP in three weeks. You now have the basics and will be able to decipher any kind of Python code that's thrown at you (you might also want to google "functional programming" with Python, but that's for another time).

Remember the basics. Classes are blueprints. We define data and methods here. The constructor is the method that is run when the object is created. We also create getter and setter methods to access and modify our object data. Class variables keep data for the class, not a particular instance of it, and deconstructors allow us to define the behavior we want to see when an object is destroyed with the del command.

Inheritance involves using a class as a base to build upon and keep all of its functionality without having to retype all of its code. Polymorphism is when things with the same name act differently depending on the type of object you have. We can do this by overriding the functions we inherit from Super classes and define our own behaviors, like when we had the cat say Meow and the dog say Bark.

Python Methodologies for Data Sciences

Other cool resources

The Python Challenge is a 33 level puzzle solving site that requires Python programming to work through the levels.

Check it out and see how far you can get!

WARNING : It's addictive... really fun stuff.

http://www.pythonchallenge.com/

Python Methodologies for Data Sciences

I know, I know...

This is where I do my mea culpas and apologize for the stuff on the syllabus that we didn’t cover. You know what? We covered it all. It’s a lot, I know, but take it slow and absorb it. Steep in it for a while. Look in the book for examples.

This brings up a good point. The syllabus says to read chapters 10 and 12. Do so, but use chapter 12 as a group of examples. You do not have to code all those programs, just read the code and follow along with it. When it comes to chapter 12 just absorb the section on OO concepts, that’s the important part there.

Python Methodologies for Data Sciences

A few simple exercises

I used to teach OOP at an enrichment school for middle schoolers. As an end of the semester project I would have them use OOP to code up a system. As an example I told them to code up a solar system or a zoo.

Think about how you would do this. If I model a solar system I would need a planet class. What kind of data does a planet have? Things like color, diameter, distance from sun, moons, etc. What kind of behaviors do they have? They rotate, they orbit the sun, they generate heat and magnetic fields in some cases, etc.

Code up a solar system with 5 planets. As data, use just name, color and diameter. For methods, have getters and setters as well as a method to override __str__ and print the planet's name. Use a class variable to keep track of the creation and destruction of your planets.