OVERVIEW

So far we have looked at three levels of Python:

1. Data Structures: for storing and retrieving information at the most basic level. These structures include Strings, Lists, Tuples and Dictionaries. Each of these is a class with associated methods (see below). For example, the List class has the append method, so that you may append new values to your list.

2. Functions: for defining a sequence of steps or actions. Functions may start by accepting arguments from the user. Functions will likely return results when they finish, by means of the return statement.

3. Classes: for encapsulating data and functionality in logical wholes. Data structures and functions, in being types you use again and again, are themselves classes. But Python allows you to extend the language to include your own types.

These three levels have a kind of hierarchical relationship: classes contain functions, which in turn contain data structures. However, since "everything is an object" in Python, these three kinds of object may occur in many configurations. For example, you might have a list of objects or a list of functions.

```
>>> def myrange(x):
     return range(1, x + 1)

>>> myrange(10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> def trinumb(n):
     return n * (n + 1) / 2

>>> trinumb(10)
55

>>> funclist = [myrange, trinumb]

>>> def runboth(n):
     for f in funclist:
          print f(n)


>>> runboth(10)
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
55
```

ABOUT CLASSES

```
class Dog:

   def  __init__(self, name):
      # constructor
      self.name = name

   def  bark(self, loud):
      if loud == 1: print 'woof'
      elif loud == 2: print 'Bark'
      elif loud == 3: print 'BARK!'
      else: print 'Huh?'

   def __repr__(self):
      # representation
      return 'Dog named %s' % self.name
```

A class consists of methods and other attributes. An object is a specific instance of a class. You may create many instances of the same class, and yet each is an individual place in memory with its own information. Every instance has a *self*.

```
>>> dog1 = Dog('Fido')
>>> dog2 = Dog('Rover')
>>> dog1
'Dog named Fido'
>>> dog2
'Dog named Rover'
```

Some methods have double-underlines in their names. These are Python's built-in method names, which are used consistently across all the class definitions. Built-in methods are typically triggered by special syntax. For example, Dog("Fido") – the class name followed by an argument in parentheses – triggers __init__, the *constructor* for the class.

When you simply enter the name of an object, and don't ask for any method or other attribute, this triggers __repr__, the method for *representing* an object.

Other methods don't have these special names. You are free to devise all kinds of methods and name them whatever you like. Notice that instance methods, unlike functions outside of classes, always have the same first argument: self. This argument is a reference to the instance itself.

However, if it’s your intention, as the programmer, to *not* have other programmers use a method directly (i.e. it’s “for internal use only”), then you may prefix the method with one or two underlines, as in _mymethod or __mymethod.

Here we are continuing to use our new Dog class:

```
>>> dog1 = Dog('Fido')
>>> dog2 = Dog('Rover')

>>> dog1
'Dog named Fido'
>>> dog2
'Dog named Rover'

>>> dog1.bark(1)
'bark'
>>> dog1.bark(3)
'BARK!'
>>> dog2.bark(4)
'Huh?'
```

PUTTING IT ALL TOGETHER

You will save your programs in modules, which are text files with the .py extension. Python comes with a large library of modules, called the Standard Library, which already do a lot for you. But the point of programming is to create your own.

When you need access to the contents of a module, either written by you, or somebody else, you simply import it. You may import everything in a module, or just specific pieces of it.

So that’s the overall structure of Python: data structures, functions and classes combine inside of modules, which you import and run.

When you run a program, you may do it interactively within the Python shell, or you may hand it off to the interpreter at the command line, or by clicking on an icon. We will look at these different ways of invoking your programs in the near future.