Contrasting Python and R

October 6, 2013

References:
Berkeley Python bootcamp (2010 version), organized by Josh Bloom

1 Programming concepts

We've covered a lot of topics in R. Part of the purpose was to learn a single language really well. But by learning a language we've also seen a lot of the core concepts that come up in other programming languages. Here are some of the topics that we've discussed:

- variable types
- passing by reference and by value
- variable scope
- the call stack
- flow control
- object-oriented programming
- matrix storage concepts
- parsing

The goals of this unit are two-fold: (1) give you a basic introduction to Python, which some of you will be using intensively in the Statistics Master's capstone course in the spring and (2) further develop understanding of programming concepts by seeing the concepts we've encountered with R in a different context.

I will note here that I am by no means a Python expert. I've picked up a bit here and there but am still learning. So, as is always the case, please do contribute your own knowledge in class, on the course Wiki on Github, and on Piazza.

Also a note on the formatting of this document. *knitr* doesn't do what I want in terms of showing the Python output (or indentation of the original code), so you'll just see the Python syntax here and we'll see the results in class.

2 Introduction to Python

Python is an interpreted language that often serves as a glue to tie together different types of code/operations in a project. It's particularly good at string manipulation and interacting with the operating system. Like R, you can run Python in a variety of ways: interactively from the command line, as a script, as a background job, and using a GUI (the *iPython notebook*). In the demos in class, we'll use the *iPython* interface, which has nice functionality including tab completion, command recall, and enhanced help information.

Like R, Python is an interpreted language and in some ways the syntax is similar.

```
print("Hi there")
print(2 + 2)
2.1 * 5  # type casting/coercion
# indentation matters
  2.1 * 5
dist = 7
dist < 6
type(2)
type(2.0)
type(2*5.0)
isinstance(dist, int)
```

Python is particularly good at string operations.

```
intro = "My name is"
name = "sam"
endLine = ".\n"
intro + name.capitalize() + endLine
intro.split(' ')
'abcdefghijkl'[0:10:2] # Python indexing starts with 0
```

The *name.capitalize()* business involves object oriented programming, which is more elegant in Python than in R.

We can do flow control in Python in similar fashion to R. Here's an *if-else* block.

```
hi = 7
if name == "samuel":
    print(intro + name.capitalize() + endLine)
elif name == "sam":
    print("My nickname is" + name.capitalize() + endLine)
else:
    print("My name is something else.\n")
```

And here's a *for* loop.

```
x = 0
for i in range(10):
    x += i

# on the command line, we need the blank line above to end the loop
print("x is " + str(x))
```

Getting help in Python

```
help(sum)
# in ipython, we can do tab completion and
#    we'll see all the methods for a given object
str = "my text"
# now type "str." and hit tab - you'll see all the
#    various string class methods. Very helpful!
```

Packages Like R, Python has a multitude of add-on packages. For a statistician, some of the most useful are *numpy* and *scipy* for scientific computing (linear algebra, optimization, integration, statistical functions, signal processing, etc.), *pandas* for Python's version of data frames, *random* for random number stuff, and *matplotlib* for plotting. There is alse *re* for regular expressions and *string* for more string operations.

```
import os  # load the os package
print(os.environ['HOME']) # use 'environ' function in the package
```

```
import numpy as np  # import and assign a new name
vec = np.random.normal(size = 1000)
vec.mean()
```

As with help on an object in iPython, we type something like `numpy.` and hit tab and see the *numpy* functions.

3 Programming concepts in Python from the R user's perspective

3.1 Variables and indexing

As in R, you do not need to declare variables before using them; i.e., Python is a dynamically typed langauge.

Python's types are different than R's but share some similarities.

List (like R lists)

```
data = 7
data = 'blue' # no problem!
data = [1, True, "same", [5, 7, 'blue']]
data[0]
data[3][0]
origData = data
data[2] = [True, False, True]
origData.extend(data)
data.sort()
```

Lists are *mutable*, meaning you can change the elements of a list. This is the case for all R objects, but not for all Python types. Lists are objects that have methods (note the *extend()* method in the example).

An aside on indexing. Indexing in Python is based on 0 as the start of the index. The indexing operators operate bit differently than R.

```
data = range(50)
data[0:3]
data[10:]
data[:5]
```

Tuple (like R lists, but immutable)

```
data = (12, True, "hi there", [5, 'sam'], (3, 11, 'Yang'))
data[0]
data[0] = 5
data[3][1] = 'sam I am'
data[4][1] = -13
data
type(data)
type(data[3])
type(data[3][0])
type(data[4])
```

Dictionary (key-value pairs - like named vectors/environments/lists in R)

```
zoo = {"lion": "Simba", "panda": None, "whale": "Moby", "numAnimals": 3}
zoo
zoo['lion']
zoo['elephants']
zoo['elephants'] = ['dumbo', 'mumbo']
zoo.keys()
```

For numerical work you'll rely heavily on the types in the *numpy* package, while for data analysis, you'll likely rely heavily on the types in the *pandas* package.

3.2 Copying in Python

Python copies are actually just references to the original object. To make an actual copy you need to work harder.

```
data = [1, 3, 5, 7]
myCopy = data
data[0] = 0
myCopy
id(data)
id(myCopy)
# how to copy?
myStaticCopy= list(data)
```

```
data[0] = Inf
myStaticCopy[0]
id(myStaticCopy)
```

3.3 Functions in Python

Python functions assign the names of the functions argument to the objects that are passed into the function. Thus we have names that are local to the function. The result is that if you pass in a mutable object and change it you affect the caller (i.e., the variable in where it was called from). However, if you overwrite a variable, that has no effect outside the function, since you're just working with the local variable. So Python acts in some ways like pass-by-value and in some ways like pass-by-reference. Here are some examples.

```
def myFun(x, y, z):
    ''' This doc string shows how to write help info for a function.
    This is my test function.
    It helps me understand how Python passes arguments. '''
    print id(x)
    print id(y)
    print id(z)
    x[0] = 7
    y = 5
    z = [3, 5]
    print id(x)
    print id(y)
    print id(z)
    return(y)

x = [0, 7, 5]
y = 3
z = [0, 1, 2]
id(x)
id(y)
id(z)
myFun(x, y, z)
x
y
```

```
z
help(myFun)
```

Let's see how Python handles scoping.

```
a = 3
def myFun(x):
    return(x*a)

myFun(7)
```

```
a = 3
def myInnerFun(x):
    return(x*a)
def myFun(x):
    a = 7
    return(myInnerFun(x))

myFun(7)
```

```
a = 3
def myFun(x):
    def myInnerFun(x):
        return(x*a)
    a = 7
    return(myInnerFun(x))

myFun(7)
```

Question: So does Python behave like R in terms of scoping, based on these examples?

Python allows arguments to be passed by name or by position, but has some more restrictive rules than R. Named arguments must be specified after positional ones in the function definition.

```
def myMult(x, y = 3, z = 5):
    return(x*y, z)

myMult(5)
myMult(5, 7)
```

```
myMult(5, 7, 11)
myMult(5, z = 4)
myMult(x = 3, z = 5, y = 7)
myMult(y = 5, x = 3, 7)

out1, out2 = myMult(5)
out3 = myMult(5)
out1
out2
out3
type(out3)
```

Summary: Python can handle arguments passed by position or by name (key word), but it cannot have arguments passed by position after named arguments. Unlike R, you can return multiple outputs, which is handy (they're actually packed up as a tuple, but as you see in the example, you can unpack it seamlessly).

3.4 Calculations and efficiency

Python has a useful functionality called *list comprehension*, which acts a bit like *lapply()* in R.

```
data = range(100)
notDiv2or3 = [num for num in data if (num % 2 != 0) and (num % 3 != 0)]
# stealing from Josh Bloom's python bootcamp (Josh is an astronomer)
particles = [{"name":" pi +" ,"mass": 139.57018},
   {"name":" pi 0" ,"mass": 134.9766},
   {"name":" eta 5" ,"mass": 47.853},
   {"name":" eta prime (958)","mass": 957.78},
   {"name":" eta c(1S)", "mass": 2980.5},
   {"name": " eta b(1S)","mass": 9388.9},
   {"name":"K+", "mass": 493.677},
   {"name":"K0" ,"mass": 497.614},
   {"name":"K0S" ,"mass": 497.614},
   {"name":"K0L" ,"mass": 497.614},
   {"name":"D+" ,"mass": 1869.62},
   {"name":"qD0" ,"mass": 1864.84},
   {"name":"D+s" ,"mass": 1968.49},
```

```
   {"name":"B+" ,"mass": 5279.15},
   {"name":"B0" ,"mass": 5279.5},
   {"name":"B0s" ,"mass": 5366.3},
   {"name":"B+c" ,"mass": 6277}]
type(particles[0])
my_mesons = [ (x['name'],x['mass'])
  for x in particles if x['mass'] <= 1000.0 and x['mass'] >= 100.0]
type(my_mesons)
type(my_mesons[0])
```

You can do vectorized operations in Python ...

```
data = range(10)
data + data
data * 3
```

but you need to use data structures from the *numPy* module.

```
import pandas
import numpy as np
vec1 = np.array([1, 2, 3])
vec2 = np.random.normal(size = 3)
vec1 + vec2
vec1 * 8
mymat = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
mymat.transpose()
mymat[0:2, :]
```

There is of course a lot more to explore here, as we did for R.

```
import timeit
expr = '''
n = 10000
x = [0]
for i in range(n):
    x = x + [i]
'''
```

```
timeit.timeit(expr, number = 10)

expr = '''
n = 10000
x2 = [0]*n
for i in range(n):
    x2[i] = i
'''

timeit.timeit(expr, number = 10)
```

Here's a little bit of code that uses pandas for Python's version of a data frame.

```
import pandas as pd
df = pd.read_table('../data/hivSequ.csv', sep = ',')
df.head()
df[0:3]
df['CD4-t0'][0:20]
```

3.5 Object-oriented programming

We've already seen that a lot of Python's basic functionality is built around objects and methods, e.g., the lengthening of a list, manipulation of strings, etc.

As we've discussed an object is a structure that groups together variables (aka attributes) and functions (aka methods) that operate on the variables in a tidy unit. The blueprint for an object is a class. Each object is an instance/instantiation/realization of a class.

Let's see how one creates a class and objects in Python.

Here's the generic syntax for a class:

```
class ClassName[(BaseClasses)]:  # can inherit from other classes

    '''[Documentation String]'''

    [Statement1] # Executed when a new instance is defined
    [Statement2]
    ...
    [Variable1] # 'Global' class variables can be defined here
```

```
    def Method1(self, args, kwargs={}):
        # Performs task 1

    def Method2(self, args, kwargs={}):
        # Performs task 2
    ...
```

Here's an example of creating a class. The *__init__* is the constructor function.

```
class indiv:
    '''The indiv class holds information about people.'''
    print("indiv class is defined.")
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def getInfo(self):
        print("Object of class 'indiv'. " + self.name + " is " + str(self.a
    def makeOlder(self):
        self.age += 1

pres = indiv("Obama", 50)
pres = indiv("Obama")
indiv
pres
pres.age
pres.makeOlder()
pres.getInfo()
```

Note that *self* needs to be an argument of every method in the class.

Copying an object is like with a list - you create a new reference but don't actually get a separate copy. To make a complete copy, do

```
import copy
newPres = copy.deepcopy(pres)
```

You can define operators for a class. E.g., if you define a *__add__* method, that would get used when '+' is called on an object in the class.

Python classes have the ability to hold information about all the objects in the class.

```
class indiv:
    '''The indiv class holds information about people.'''
    population = 0
    print("indiv class is defined.")
    def __init__(self, name, age):
        self.name = name
        self.age = age
        indiv.population += 1
    def getInfo(self):
        print("Object of class 'indiv'. " + self.name + " is " + str(self.a

pres = indiv("Obama", 50)
friend = indiv("Fang", 40)
indiv
pres
pres.getInfo()
indiv.getInfo(pres)  # alternative
```