# Learning R via Python
# ...or the other way around

Drew Conway

Dept. of Politics - NYU

January 7, 2010

## What We'll Cover

Brief review of Python

- The Zen of Python
- How are R and Python the same, and how are they different

Similar Data Structures

- Python dictionary
- R list

Example - Testing if an integer is prime

- Create a function and returning a value
- Using while-loops

Bridging the gap with RPy2

- Running a regression with R inside Python

Python resources

# Fundamental Elements of Python

## The Python coder's creed

```
>>> import this
The Zen of Python, by Tim Peters

Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex.
Complex is better than complicated.
Flat is better than nested.
Sparse is better than dense.
Readability counts.
Special cases aren't special enough to break the rules.
Although practicality beats purity.
Errors should never pass silently.
Unless explicitly silenced.
In the face of ambiguity, refuse the temptation to guess.
There should be one-- and preferably only one --obvious way to do it.
Although that way may not be obvious at first unless you're Dutch.
Now is better than never.
Although never is often better than *right* now.
If the implementation is hard to explain, it's a bad idea.
If the implementation is easy to explain, it may be a good idea.
Namespaces are one honking great idea -- let's do more of those!
```

# How are R and Python alike, and how are they different?

Similarities

Differences

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

### Array of squared values

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
```

## Differences

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

### Array of squared values

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

## Differences

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

**Array of squared values**

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

## Differences

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

**Array of squared values**

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

**Array of squared values**

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

General purpose OOP vs. Functional OOP

- Python is a high-level language for application development
- R is a collection of functions for data analysis

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

**Array of squared values**

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

General purpose OOP vs. Functional OOP

- Python is a high-level language for application development
- R is a collection of functions for data analysis

Syntax structure

- Python syntax forces readability through whitespace
- R emphasizes parsimony and nesting

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

**Array of squared values**

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

General purpose OOP vs. Functional OOP

- Python is a high-level language for application development
- R is a collection of functions for data analysis

Syntax structure

- Python syntax forces readability through whitespace
- R emphasizes parsimony and nesting

**Creating a function**

```
# In Python functions are declared
>>> def my_func(x):
...
```

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

### Array of squared values

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

General purpose OOP vs. Functional OOP

- Python is a high-level language for application development
- R is a collection of functions for data analysis

Syntax structure

- Python syntax forces readability through whitespace
- R emphasizes parsimony and nesting

### Creating a function

```
# In Python functions are declared
>>> def my_func(x):
...
# In R functions are assigned
> my.func<-function(x) { ... }
```

# How are R and Python alike, and how are they different?

## Similarities

Functional programming paradigm

### Array of squared values

```
# In Python we use a lambda nested in map
>>> map(lambda x: x**2,range(1,11))
[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]
# In R we define a function in sapply
> sapply(1:10,function(x){return(x**2)})
 [1]   1   4   9  16  25  36  49  64  81 100
```

Object-oriented programming

- Both languages support robust class structures
- In R, there are the S3 and S4 classes

Easily call C/Fortran code

- Due to their high-level, both languages provide functionality to call compiled code from lower-level languages

## Differences

General purpose OOP vs. Functional OOP

- Python is a high-level language for application development
- R is a collection of functions for data analysis

Syntax structure

- Python syntax forces readability through whitespace
- R emphasizes parsimony and nesting

### Creating a function

```
# In Python functions are declared
>>> def my_func(x):
...
# In R functions are assigned
> my.func<-function(x) { ... }
```

For more info see: StackOverlow.com discussion on topic

## Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

# Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

## Working with the dict type

```
# Keys and values can be of any data type
>>> fruit_dict={"apple":1,"orange":[0.23,0.11],"banana":True }
```

# Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

## Working with the dict type

```
# Keys and values can be of any data type
>>> fruit_dict={"apple":1,"orange":[0.23,0.11],"banana":True }

# Can retrieve the keys and values as Python lists (vector)
>>> fruit_dict.keys()
["orange","apple","banana"]
```

# Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

## Working with the dict type

```
# Keys and values can be of any data type
>>> fruit_dict={"apple":1,"orange":[0.23,0.11],"banana":True }

# Can retrieve the keys and values as Python lists (vector)
>>> fruit_dict.keys()
["orange","apple","banana"]

# Or create a (key,value) tuple
>>> fruit_dict.items()
[("orange",[0.23,0.11]),("apple",1),("Banana",True)]
```

# Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

## Working with the dict type

```
# Keys and values can be of any data type
>>> fruit_dict={"apple":1,"orange":[0.23,0.11],"banana":True }

# Can retrieve the keys and values as Python lists (vector)
>>> fruit_dict.keys()
["orange","apple","banana"]

# Or create a (key,value) tuple
>>> fruit_dict.items()
[("orange",[0.23,0.11]),("apple",1),("Banana",True)]
# This becomes especially useful when you master Python ``list comprehension''
```

# Python Dictionaries

In Python the dict type is a data structure that represents a key→value mapping

## Working with the dict type

```
# Keys and values can be of any data type
>>> fruit_dict={"apple":1,"orange":[0.23,0.11],"banana":True }

# Can retrieve the keys and values as Python lists (vector)
>>> fruit_dict.keys()
["orange","apple","banana"]

# Or create a (key,value) tuple
>>> fruit_dict.items()
[("orange",[0.23,0.11]),("apple",1),("Banana",True)]
# This becomes especially useful when you master Python ``list comprehension''
```

The Python dictionary is an extremely flexible and useful data structure, making it one of the primary advantages of Python over other languages

- Luckily, there is a fairly direct mapping between the Python dict and R lists!

# R Lists

Similarly to the Python dictionary, the R list is a key$\rightarrow$value mapping

# R Lists

Similarly to the Python dictionary, the R list is a key→value mapping

## Working with an R list

```
> fruit.list<-list("apple"=1,"orange"=c(0.23,0.11),"banana"=TRUE)
> fruit.list
$apple
[1] 1

$orange
[1] 0.55 0.11

$banana
[1] TRUE
```

# R Lists

Similarly to the Python dictionary, the R list is a key→value mapping

## Working with an R list

```
> fruit.list<-list("apple"=1,"orange"=c(0.23,0.11),"banana"=TRUE)
> fruit.list
$apple
[1] 1

$orange
[1] 0.55 0.11

$banana
[1] TRUE
```

Notice, however, that R will always treat the value of the key/value pair as a vector; unlike Python, which does not care the value's data type. Of the many R programming paradigms, the "vectorization of all data" is among the strongest.

# R Lists

Similarly to the Python dictionary, the R list is a key→value mapping

## Working with an R list

```
> fruit.list<-list("apple"=1,"orange"=c(0.23,0.11),"banana"=TRUE)
> fruit.list
$apple
[1] 1

$orange
[1] 0.55 0.11

$banana
[1] TRUE
```

Notice, however, that R will always treat the value of the key/value pair as a vector; unlike Python, which does not care the value's data type. Of the many R programming paradigms, the "vectorization of all data" is among the strongest.

## Using unlist

```
> unlist(fruist.list)
apple orange1 orange2  banana
 1.00    0.55    0.11    1.00
```

## Testing a Prime in Python

We are interested in testing whether integer $X$ is prime, and to do so we will:

- Declare a function
- Use a while-loop
- Return a boolean

# Testing a Prime in Python

We are interested in testing whether integer $X$ is prime, and to do so we will:

- Declare a function
- Use a while-loop
- Return a boolean

**The is_prime function**

```
def is_prime(x):
    if x%2 == 0:
        return False
    else:
        y=3
        while y<x:
            if x%y==0:
                return False
            else:
                y+=2
        return True
```

# Testing a Prime in Python

We are interested in testing whether integer $X$ is prime, and to do so we will:

- Declare a function
- Use a while-loop
- Return a boolean

**The is_prime function**

```
def is_prime(x):
    if x%2 == 0:
        return False
    else:
        y=3
        while y<x:
            if x%y==0:
                return False
            else:
                y+=2
        return True
```

Notice the number of lines in this simple function. In contrast to R, and many other programming languages, Python's whitespace and indentation requirements force verbose code; however, the end result is very readable.

# Testing a Prime in R

Unlike Python where functions are declared explicitly, R creates functions through assignment. In many ways, this is somewhere between the Python lambda function and an explicit function declaration.

# Testing a Prime in R

Unlike Python where functions are declared explicitly, R creates functions through assignment. In many ways, this is somewhere between the Python lambda function and an explicit function declaration.

## The is.prime function

```
is.prime<-function(x) {
    if(x%%2==0) {return(FALSE)
    } else {
        y<-3
        while(y<x) ifelse(x%%y==0,return(FALSE),y<-y+2) }
    return(TRUE)
}
```

# Testing a Prime in R

Unlike Python where functions are declared explicitly, R creates functions through assignment. In many ways, this is somewhere between the Python lambda function and an explicit function declaration.

### The is.prime function

```
is.prime<-function(x) {
    if(x%%2==0) {return(FALSE)
    } else {
        y<-3
        while(y<x) ifelse(x%%y==0,return(FALSE),y<-y+2) }
    return(TRUE)
}
```

Contrast the width of this version of the function with the length of the previous

- With the `ifelse` function R allows you to compress many lines of code
- A good example of the difference between a largely function programming language with objects (R) vs. a largely objected-oriented language with a core set of functions (Python)

## Bridging the Gap with RPy2

Python is widely adopted within the scientific and data communities

- The combination of SciPy/NumPy/matplotlib represents a powerful set of tools for scientific computing
- Companies like Enthought are focused on servicing this area

# Bridging the Gap with RPy2

Python is widely adopted within the scientific and data communities

- The combination of SciPy/NumPy/matplotlib represents a powerful set of tools for scientific computing
- Companies like Enthought are focused on servicing this area

There are, however, many times when even this combination cannot match the analytical power of R—enter `RPy2`

- RPy2 is a Python package that allows you to call R directly from within a Python script
- This is particularly useful if you want to use one of R's base functions inside Python

# Bridging the Gap with RPy2

Python is widely adopted within the scientific and data communities

- The combination of SciPy/NumPy/matplotlib represents a powerful set of tools for scientific computing
- Companies like Enthought are focused on servicing this area

There are, however, many times when even this combination cannot match the analytical power of R—enter `RPy2`

- RPy2 is a Python package that allows you to call R directly from within a Python script
- This is particularly useful if you want to use one of R's base functions inside Python

For example, creating a function to mimic R's `lm` in Python—even using the above combination—would be very time consuming. To avoid this, we will use RPy2 to call `lm` from within a Python script and plot the data.

# Bridging the Gap with RPy2

Python is widely adopted within the scientific and data communities

- The combination of SciPy/NumPy/matplotlib represents a powerful set of tools for scientific computing
- Companies like Enthought are focused on servicing this area

There are, however, many times when even this combination cannot match the analytical power of R—enter `RPy2`

- RPy2 is a Python package that allows you to call R directly from within a Python script
- This is particularly useful if you want to use one of R's base functions inside Python

For example, creating a function to mimic R's `lm` in Python—even using the above combination—would be very time consuming. To avoid this, we will use RPy2 to call `lm` from within a Python script and plot the data.

Disclaimer: the syntax can be a bit tricky to understand at first, so allow for a fairly steep learning curve!

# Calling `lm` from Python with `RPy2`

## Creating a linear model in Python

# Calling lm from Python with RPy2

## Creating a linear model in Python

```
# First import RPy2 and create an instance of robjects
import rpy2.robjects as robjects

r = robjects.r
```

# Calling lm from Python with RPy2

## Creating a linear model in Python

```
# First import RPy2 and create an instance of robjects
import rpy2.robjects as robjects

r = robjects.r
# Create the data by passing a Python list to RPy2, which interprets as an R vector
ctl = robjects.FloatVector([4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14])
trt = robjects.FloatVector([4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69])
group = r.gl(2, 10, 20, labels = ["Ctl","Trt"])
weight = ctl + trt
```

# Calling lm from Python with RPy2

## Creating a linear model in Python

```
# First import RPy2 and create an instance of robjects
import rpy2.robjects as robjects

r = robjects.r
# Create the data by passing a Python list to RPy2, which interprets as an R vector
ctl = robjects.FloatVector([4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14])
trt = robjects.FloatVector([4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69])
group = r.gl(2, 10, 20, labels = ["Ctl","Trt"])
weight = ctl + trt
# RPy2 uses Python dictionary types to index data
robjects.globalEnv["weight"] = weight
robjects.globalEnv["group"] = group
```

# Calling lm from Python with RPy2

## Creating a linear model in Python

```
# First import RPy2 and create an instance of robjects
import rpy2.robjects as robjects

r = robjects.r
# Create the data by passing a Python list to RPy2, which interprets as an R vector
ctl = robjects.FloatVector([4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14])
trt = robjects.FloatVector([4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69])
group = r.gl(2, 10, 20, labels = ["Ctl","Trt"])
weight = ctl + trt
# RPy2 uses Python dictionary types to index data
robjects.globalEnv["weight"] = weight
robjects.globalEnv["group"] = group
# Run the models
lm_D9 = r.lm("weight ~ group")
print(r.anova(lm_D9))

lm_D90 = r.lm("weight ~ group - 1")
print(r.summary(lm_D90))
```

# Calling lm from Python with RPy2

## Creating a linear model in Python

```
# First import RPy2 and create an instance of robjects
import rpy2.robjects as robjects

r = robjects.r
# Create the data by passing a Python list to RPy2, which interprets as an R vector
ctl = robjects.FloatVector([4.17,5.58,5.18,6.11,4.50,4.61,5.17,4.53,5.33,5.14])
trt = robjects.FloatVector([4.81,4.17,4.41,3.59,5.87,3.83,6.03,4.89,4.32,4.69])
group = r.gl(2, 10, 20, labels = ["Ctl","Trt"])
weight = ctl + trt
# RPy2 uses Python dictionary types to index data
robjects.globalEnv["weight"] = weight
robjects.globalEnv["group"] = group
# Run the models
lm_D9 = r.lm("weight ~ group")
print(r.anova(lm_D9))

lm_D90 = r.lm("weight ~ group - 1")
print(r.summary(lm_D90))
```

For more info check out the RPy2 documentation (from where this example was stolen!)

# Python Resources

Where to get Python

- Binaries for several operating systems and chipsets can be downloaded at `http://www.python.org/download/`
- For OS X and Linux some version of Python comes pre-installed
- Some controversy over what version to use

How to use Python

- Several very good development environments, once again SO has this covered
- My opinion, OS X: TextMate and Windows: Eclipse with PyDev, Linux: n/a

How to Learn Python

- Several online resources: Dive into Python and Python Rocks
- For scientific computing: Enthought Webinars
- For applications to finance: NYC Financial Python User Group

# Python Resources

Where to get Python

- Binaries for several operating systems and chipsets can be downloaded at `http://www.python.org/download/`
- For OS X and Linux some version of Python comes pre-installed
- Some controversy over what version to use

How to use Python

- Several very good development environments, once again SO has this covered
- My opinion, OS X: TextMate and Windows: Eclipse with PyDev, Linux: n/a

How to Learn Python

- Several online resources: Dive into Python and Python Rocks
- For scientific computing: Enthought Webinars
- For applications to finance: NYC Financial Python User Group

The best way to learn any language is to have a problem and solve it using that language!