R to Python

Contents

```
install.packages("bookdown","reticulate")
# or the development version
# devtools::install_github("rstudio/bookdown")
```

Chapter 1

Front Material

1.1 Colophon

R to Python

Thoughts of an R programmer Learning Python

by David A York

http://crunches-data.appspot.com/contact.html

self-published, on Toth-York Imprint, Calhoun GA USA, November 27, 2018

https://github.com/medmatix/RtoPythonThoughts

> "In ancient Roman religion and myth, Janus (/ d e nəs/; Latin: IANVS (Iānus), pronounced [ja .nus]) is the god of beginnings, gates, transitions, time, duality, doorways,[1] passages, and endings. He is usually depicted as having two faces, since he looks to the future and to the past."
>
> from Wikipedia, https://en.wikipedia.org/wiki/Janus, accessed 27 November, 2018 at 4:17PM

1.2 Preface

This book is a work in progress on the thoughts of an R programmer (the author) moving to (or learning) Python. It is intended as a comparison of R and Python for those already using R. The hop is to ease the way for those in particular newer to programming as opposed to the quick task oriented scripting which R lends itself so well to.

As with any language the most visible presence on line are the hardened champions of the languages with the pressure to standardization (not a bad thing per-se) and dogmatic adherence to the "blah blah" way of doing things (which is desirable I think).

Having started as an R user, then as an R programmer, I have relatively recently embarked on learning python as well. Why I would do this is rather complicated to explain, particularly as I had knowledge of other general programming languages, which would conceivably serve such a role, particularly BASIC and Java. Suffice to say that as a budding Data Scientist I see lots of reasons to know both R and Python.

Figure 1.1: Janus

BASIC, though still extant it is very limited in support and relevant libraries by comparison. Java . . . well, as a strong open source advocate I feel Java is bound to Oracle in many ways for all time. It is likely they have no intention of truly releasing it into the public domain, ever. There is far more rationel to learn C++ these days than Java for the sake of what is arguably only minor difference in learning curve.

I absolutely love R! When I returned to school to study Applied Math and Statistics it was an ever present partner for me. It could do most things I needed, though matrix math was less smooth than Matlab the price was right (not withstanding Octave or Scilab). However, I increasingly found the need for a general purpose programming language and Python was there growing in the guise of 'the' data science alternative.

But while R was great with vectorized functions python, at it's core, was weak in this regard. The libraries for python have been growing steadily in number and variety, and the functionality gap between it and R narrows in cluding for vectorized application. I still cannot see a Data Scientist being as productive not knowing R at this point. However, I see no reason not to also know Python - ergo, I think any serious Data Science needs to know both; AND, there is the added incentive of the interoperability of R and Python (and C++)! From R we have XRPython and reticulate and from Python we have Rpy2.

The Organization of the book is in 2 parts. A close Comparison and discussion of R and Python unburdened with a mandate to teach beginning programming in either. The is overview of part 1, I only touch quite generally on the nature of the usual programming sturctures. Variables, datatypes and data structures, binary decisions, repeating code blocks and so are part of all declarative or functional programming language - the rest is just details. The Same can be side for laguages introducing object-oriented programming models.

In the second part of the book, an opportunity is taken to compare how data science is done in R and Python in the various domains we work.

One could consider this as a second source for R users learning python, after the beginner courses to get the python syntax vocabulary.

Chapter 2

Introduction

Learning a new programming or scripting language begs comparison. Often this is not overly helpful to the learner who, coming from a zone of comfort is often frustrated by old subtleties not consciously recognized, being yet unknown in the new. I recall a family member, completely new to computers, saying why doesn't it know what I mean expecting the implied parts of language to transfer from English (or German, or Chinese etc.) to computer language.

R writers early on recognize the absence of vectorization of functions in core python. It's available though in numpy and pandas, but base python doesn't have it and they miss it. You'll get there, really. Object Orientation is also not natural to either R or Python. It came later to both, but it seems more smoothly accomplished by python than R.

I soon realized, though that you had to know the programming commonalities of python first; R comparisons really should start with the libraries of python. R in someways has cut right to the data science chase often at the expense of strong programming constructs. This is the nod to interactive coding and scripting rather than the need to undertake formal programming. Python began early on with a scripting functionality but it wasn't until the development of iPython the interactive python could be realized in the way that R users were used to.

All this is not to say that R was lacking in basic programming constructs from the start, it wasn't; but he proportion of users who used R in a quick-and-dirty small problem focused way was very high. This suited it's use in a statistical learning environment. I should think that those who try to start python at the same time as their first statistics course will find the learning curve too steep to help with the statistics part of the need.

Part I

R and Python, Side by Side

Chapter 3

A Comparative Look

As already mentioned, it is not the intent of this treatise to teach either beginning R or python scripting. There are many good sources for this and yet another beginners book or tutorial is, I believe, not needed. For ease of reference I include Appendix1 as an overview of relevant R and python syntax for the general programming constructs any functional language. In Part 1, at the risk of being pedantic, I am simply giviing a fairly generic overview of the components of the typical script or program as a framework to be used in organizing our R and Python comparisons.

3.1 Functional Programming Commonalities.

In functional programming operational code is incorporated by task and purpose into blocks of code which are called by name in the body of a script. This usually includes some connecting statemts and assignments which cause the functions to operate together to accomplish some larger task.

Historically, Fortran and Basic code in common accademic and educational use, became unwieldy as users became, more and more, writers of rampant spagetti code.The introduction of the concept of 'structured programming' and the uniform reliance on subroutine and function call tamed the spagetti-monster. Extension to functional programming with pascal and modern C was a natural extension of this trend.

So the reliance of R and Python on script collections of functions calling each other is what we are at now. Functions are subprograms that have all the statement tasks common to most programs. I will deal in a general way with these statement types and include comparsions between how R and Python each express the tasks. It is not the intent of this book to actually teach beginning programming as previously mentioned. Consult the bibliography for some suggested sources for peginning programmers.

The later innovation of object oriented programming concepts evolving from this will be touched on in a comparative fashion in another chapter.

3.1.1 Statements and Expressions

A program or script is of course considered as a series of statement stored together. These statements constitute expressions of mathemetical equations (see specifically Chapter 5), choices, and other orders to be carried out by the machine.

With any program or script statements tasks and values stored are executed or accessed from memory locations. This was the innovation conceived of by John Von Neumann which moved programming from the moving wire jumpers to the input to the machine from various codes punched into tapes or cards to be held in memory while being used.

3.1.2 Assignment of Values to Memory and the Variable

Variables are memory locations assigned values as the computing process goes on. The classification of specific locations determines a type of data held there.

The codes stored in the machine include a specific operator code to direct the move of a value received for input or a statement's result to a memory location. This is called and assignment.

Traditionally R has used the combined dash-less-than symbols '->' for this purpose but the simple equals sign '=' now also works in this fashion. Python uses the same equals sign for assignments.

Variables are named in both R and Python and neither may begin with a digit. There are other restrictions and conventions which are not gone into here.

3.1.3 Decisions and Choices

As currently implemented all choices by a computer resolve to yes or no results. Some semblance or the greys we think in are a result of a collection or range of yes-no questions. These choices are explicit of implicit 'if' queries.

3.1.4 Doing it Over Again (and again ...)

Loops are created within a program logic, usually involving a choice test to be executed which cause tasks to be repeated zero or more times. It is considered bad parctice to create a loop without some form of exit (short of having to power down the machine).

3.1.5 Functions

Functions in fortran were short program clips that did a task and immediately returned a value . Subroutines are more like what functions in R and Python are, though returning to the calling place with some value is a frequent option with these functions as well. Functions may provide a new programming element like a square root (python, math.sqrt(x)) extending the language, or functions may carry out a whole task like carring out a liner regression, storing objects of the answer in memory for retrieval, or return as an object explicitly. Ofter a function is called that builds a whole grphic plot for display or other output when called.

In R,

```
function.name <- function(arguments)
{
  computations on the arguments
  maybe other code too
}
```

and in Python indents take the place of brackets to define code blocks. Self in the function spec is the calling code block, and if omitted, is created by python implicitly. So, your first argument one way of another is the calling namespace. Self in the indented code block is the function itself.

```
def aroutine(self, args)
  computations on args
  maybe other code too
```

The notation of R is C (and Java) - like and Python is Pascal-like in format. Unlike pascal there is no 'begin:' keyword required at anytime in the script.

Functional programming with R and Python, will be expanded on in it's own chapter later on.

3.1.6 Objects and Classes

In general terms we will come to consider all program elements as objects. A Class will be considered as a pattern definition for an object. All objects are used as instances of some previously defined class.

Chapter 4

Set-up for R and Python Exercises

Clearly you have to get R and python onto your system. I have a bias about depending on any en block installed, like Anaconda etc. Anaconda is convenient but is too easy to loose locations of programs and tools and get conflicts with installs outside of the grouped installers like different Python destibutions and difficult to manage path complexity.

Just the same, Anaconda is a good choice if you are doing a clean install and intend to do everthing inside that framework.Then installing Anaconda,Python Data Science Platform can make things much easier in the long run for integrating R and Python and Reproducible Research work.

For my own purposes, I install each separately. Start with R, and then install Python through the Next Notebook.Finally install .

All of these sites will help you approach installs and configurations in a variety of ways. As an R programmer first I installed R, then RStudio. Once installed, RStudio is a convenient way to install the R packages as well and then Python 3 atPython Software Foundation. From that point on use PIP package manager for python and RGUI for package installs for python and R respectively and install jupyter see Project Jupyter. This gave the maximum flexibility for R and Python in their own environments.

I did do a parallel Anaconda install with R and RStudio and iPython for Jupyter Notebooks but as mentioned there was some path confusion with this approach. It does set you up nicely for any approach to data science and development you could choose down the road.

You really are going to need an IDE for Python. Spyder, available separately or installed through Anaconda is a solid python focused choice. I use Eclipse as you can use it for multiple languages including R (not as strong yet) and python.

4.1 Package Libraries

The strength of both R and Python is from their libraries. CRAN Package Repository and Bioconnector are both well stocked with general and special purposed functions for R. RStudio develops R Packages integrating well with the RStudio environment. The are lost of R packages in development awaiting addition to CRAN or hosted outside of CRAN such as on Github.

If the reader has spent any time with R at all you have likely made inroads into these package sources. So how does Python compare? Python comes with a large standard library providing “out of the gate” functionality to compare with R-core. The best handing of the intricacies of the python standard library I’ve found is Doug Helmann’s The Python 3 Standard Library by example. Python Software foundation hosts the Python Package Index , PyPi analygous to the CRAN repositories and Bioconnector. Also, the most important resource to get you up to speed is SciPy tools consisting of SciPy, Numpy, Pandas and Matplotlib.

These are the essential minimum to do any R-like work in python. These and more can be accessed as well through the Numfocus Projects. Finally, you have to check out the StatsModels package site.

4.2 Pulling it Together

Having apprised ourselves of the breadth and availability of these resources we will go forward and organize some of the ideas. R packages can be installed with the R GUI but the RStudio environment is much easier for this. Once a package is installed it is available in all environments I've alluded to. Compiling and installing is always best and I thing RStudio makes this fairly painless. Remember to start the GUI, Anaconda or RStudion environment as administrator to keep things together.

4.2.1 The Prompts

When accessing the R and Python interpreters from the command shell (or bash) prompt,

```
$
```

after the version and copyright banners, if present, you are at an interpreter prompt. For R this is:

```
>
```

for python,

```
>>>
```

for iPython

```
In [1]:
```

from this point you enter what ever commands or statements you wish to execute. These are executed immediately, or continued with continuation prompts until executable. Once executed any declarations and output are retained if the were assigned to some object to be held in memory. Immediate answers are generally lost.

Otherwise, one groups statements into a file to be read all at once and executed as a group. Objects created are again retained in memory by default. This is a script of program. Script file extensions are standardized for the operating system to recognize, eg.(.R for R, and .py or pyc for python). Python scripts intended to be called by other scripts for library code to be incorporated are called modules and end in the same .py extension. A package can be considered, in both R and python) as a collection of declarations, functions, classes, or modules for library purposes.

4.2.2 Installing Packages

From the R GUI prompt we have,

```
package-install()
```

Python packages from any source mentioned above can be installed with the PIP utility. This can be done from the shell prompt (remember to start as administrator or root):

```
$ pip install matplotlib
  # OR
$ python -m pip install matplotlib
```

This can be done as root with the anaconda prompt as well.

4.2.3 Using Packages

In R we must make sure the package (once installed) is loaded for our session. This is done with the library() function,

```
library("reticulate")
```

while in python we import the package previously installed as,

```
import numpy
```

Once imported or loaded into either, we can access the functions of MASS directly by name. If we have a previous function of the same name it gets overwritten unless in python we keep the name spaces separate. This can be done by the form of the import statement. We can also import only slected functions from a package module.

```
import numpy as np             # usual format to use namespace segregation of functions
from numpy import sum, matrix  # economical import, still can conflick without use of as
```

4.2.4 Using R and Python Together

There are several ways that one could find helpful to use R and Python together, to take best advantage of their respective strengths

4.2.4.1 Working in R and Calling Python

In this instance one would find working in R the necessary starting place and find the need to employ python functions or packages helpful to your needs. A call would be made to the python function from R to have a task completed by python and control returned to the R program.

```
cat("PI in R is", pi)
```

```
## PI in R is 3.141593
```

```
import math
print(math.pi)
```

```
## 3.141592653589793
```

4.2.4.2 Working in Python and Calling R

Working in python and calling an R object would look like,

```
# Accessing an R object from from python
# (example from: http://rpy.sourceforge.net/rpy2/doc-dev/html/introduction.html)
import rpy2.robjects as robjects
rpi = robjects.r['pi']
print("PI from R = ", rpi)
```

4.2.4.3 Working in Jupyter and Writing in R, Python or Both in the Same Notebook

There are multiple kernel plugins available for jupyter notebooks. As A rule, the notebooks only use one language kernel at a time. An excepti9on is the SOS kernel.

4.2.4.4 Working in rmarkdown and Using R and Python Code Chunks

Note: I have found behavior is unpredictible to call R objects back with Python chunks run in rmarkdown (ie. inside another instance of R). The identity of R_User for rpy2 gets lost. There are limites to how deeply you should nest languages within languages.

That said, the rmarkdown chuncks look like this.

````
'''{r}
cat("PI in R is", pi)
'''

'''{python}
import math
print(math.pi)
'''
````

Chapter 5

Basic Mathematics in R and Python

This is pretty straight forward and intuitive stuff. Just the same, for completeness here it is. The Standard libraries of python have the function beyond the four arithmetic operations.

5.1 Add, Subtract, Multiply and Divide

The usual quick calculation would be done in immediate or interactive mode without relying on print statements. There is outwardly no difference if the print is used as would be the case in a script. However Printing a set of values reqires the set be organized into an object in itself, a vector.

```
# Loading reticulate package to bring python interpreter on line.
library("reticulate")
```

5.1.1 R Scripting

```
# Some immediate calculations, ie. using R like a calculator
2 + 3           # (simple interactive) addition
```

```
## [1] 5
```

```
5 - 2           # subtraction
```

```
## [1] 3
```

```
6 * 2           # multiplication
```

```
## [1] 12
```

```
4 / 3           # division
```

```
## [1] 1.333333
```

```
print(4 / 3)    # calculation in a script requires print to display an answer
```

```
## [1] 1.333333
```

```
# Variable assignment and printing grouped variables as a vector
a <- (2 + 3)           # addition
b <- (5 - 2)           # subtraction
c <- (6 * 2)           # multiplication
```

```
d <- (4 / 3)            # division
print (c(a, b, c, d))   # printing all results together one statement
```

```
## [1]  5.000000  3.000000 12.000000  1.333333
```

Python, on the otherhand considers literal expressions and variables individually (though we'll see they can be groups as well.) This thr puthon print statement is by default printing one or more individual objects as scalars whereas R print function prints object and there is no scalar. The simplest object in R is the vector.

5.1.2 Python Scripting

The simplest object in python is the scalar, having one of five **basic datatypes**,

- Integers
- Floating-Point Numbers
- Complex Numbers
- Strings
- Boolean Type

```
# The same immediate calculations, ie. using iPython like a calculator
print(2 + 5, 3 * 8, 5 - 1, 67 / 3) # printing all immediate results together, one statement
```

```
## 7 24 4 22.333333333333332
```

```
# OR assignment and print
a = 2 + 5           # assigned addition
b = 5 - 1           # assigned subtraction
c = 3 * 8           # assigned multiplication
d = 67 / 3          # assigned division
print(a,b,c,d)      # printing all results together one statement
```

```
## 7 4 24 22.333333333333332
```

```
print('a =', a, 'b =',b, 'c =',c, 'd =',d) # annotating print is just using 4 literals and 4 variables
```

```
## a = 7 b = 4 c = 24 d = 22.333333333333332
```

This differs from R where a single object is printed but that object may be a group of objects combined.

5.2 Other basic algebraic operators in R and Python

Next lets compare modulus, powers and interger division in R and python,

```
35 %% 2     # modulo operator
```

```
## [1] 1
```

```
35 %/% 2    # integer division
```

```
## [1] 17
```

```
35^2        # powers in R\
```

```
## [1] 1225
```

```
print(35%2)       # modulo operator
```

```
## 1
```

```
print(35 // 2)    # integer division       * note difference from R
```

```
## 17
```

```
print(35**2)      # powers in python     * note difference from R
```

```
## 1225
```

Note here with python the print statements are required otherwise only the last immediate calculation would get output to text.

When we extend our algebraic calculations in R the usual functions of square root, absolute value and exponentiation we still haven't had to load any library packages. All these are part of the core R.

```
abs(-5)          # absolute value
```

```
## [1] 5
```

```
sqrt(16)         # square root function
```

```
## [1] 4
```

```
exp(0)           # exponent (e^0 etc)
```

```
## [1] 1
```

Not necessarily so for python. We still have the basic functions for algebraic tasks but from sqrt on ward, we now need to go to the standard math library for these. As an aside we could have got more powerful versions of these functions (and more) importing numpy or numba or scipy library packages. This will come up in more detail later.

```
print(abs(-5))
```

```
## 5
```

```
print(divmod(10,3))             # Returns quotient and remainder of integer division
```

```
## (3, 1)
```

```
print(max(2,10,3,14,4,28,7))    # Returns the largest of given arguments or items in an iterable
```

```
## 28
```

```
print(min(9,2,10,3,14,4,28,7))  # Returns the smallest of the given arguments or items in an) iterable
```

```
## 2
```

```
print(pow(3,4))                   # Raises a number to a power
```

```
## 81
```

```
print(round(3.1415962,3))         # Rounds a floating-point value
```

```
## 3.142
```

```
print(sum([2,3,4,5,6]))
```

```
## 20
```

```
import math                     # Import math package/module from Standard Library
print(math.sqrt(16))            # square root function
```

```
## 4.0
```

```
print(math.exp(0))              # exponentiation function (powers of e)
```

```
## 1.0
```

```
print(math.log10(5))            # base 10 logarithm
```

```
## 0.6989700043360189
```

```
print(math.log(5))              # natural logarithm
```

```
## 1.6094379124341003
```

A full list of the built-in (as opposed to Standard Library Funtions) is displayed, from the python documentation, below.

5.2.0.1 Python Built-in Functions

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	**import**()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

R has a very large set of built-in functions available before the need to load any libraries arises. The R Language Reference provides a comprehensive background on built in fucntions as well as the rest of the language.

So R is centered around the mandate of being a complete mathematical system, much like Matlab,(Octave or SciLab) which Python's manadate is that of a full featured scripting and programming environment.

However, calling standard library modules in python puts it quite easily and effectively on a even par with R. Going into more complex activites requires thar both load or import fucther libraries. Like **Python's Standard Library**, **the R Core Library** shipping with the R install is described in the Full R Reference Manaual and covers a wide range of mathematical, statistical and programming solutions.

We will go deeper into these and the other external Libraries and modules in the next chapter, covering functional programming in more detail. As well focused discussions on specific Data Science domains form a block of Chapters in Part II of the book.

Chapter 6

Functional Programming with R and Python

At this point we are ready to discuss the functional programming paradigm in R and Python. The other important paradigm, Object-Oriented, will be discussed in the following Chapter.

6.1 Defining Functions

6.1.1 R Scripting

6.1.2 Python Scripting

6.2 Calling and Using Functions

6.2.1 R Scripting

6.2.2 Python Scripting

6.3 The Core or Standard Libraries

6.3.1 R Scripting

6.3.2 Python Scripting

Part II

Data Science Topics in Python Compared to R

Chapter 7

Using Probability Distributions with R and Python

At this point we are ready to discuss the functional programming paradigm in R and Python. The other important paradigm, Object-Oriented, will be discussed in the following Chapter.

7.1 Basic Probability Issues

7.1.1 R Scripting

7.1.2 Python Scripting

7.2 Using the Distrbutions

7.2.1 R Scripting

7.2.2 Python Scripting

7.3 Other Libraries with Probability and Statistical Packages

7.3.1 R Scripting

7.3.2 Python Scripting

Chapter 8

Descriptive Statistics and Data Exploration

At this point we are ready to discuss the functional programming paradigm in R and Python. The other important paradigm, Object-Oriented, will be discussed in the following Chapter.

8.1 Defining Functions

8.1.1 R Scripting

8.1.2 Python Scripting

8.2 Calling and Using Functions

8.2.1 R Scripting

8.2.2 Python Scripting

8.3 The Core or Standard Libraries

8.3.1 R Scripting

8.3.2 Python Scripting

Chapter 9

Statistical Analysis and Modeling

At this point we are ready to discuss the functional programming paradigm in R and Python. The other important paradigm, Object-Oriented, will be discussed in the following Chapter.

9.1 Defining the Available Functions

9.1.1 R Scripting

9.1.2 Python Scripting

9.2 Calling and Using Functions

9.2.1 R Scripting

9.2.2 Python Scripting

Chapter 10

Non-Stochastic Models

Many mathematical models are deterministic. This includes mathematical programming.

Chapter 11

Object-Oriented Programming with R and Python

At this point we are ready to discuss the functional programming paradigm in R and Python. The other important paradigm, Object-Oriented, will be discussed in the following Chapter.

11.1 Defining Classes

11.1.1 R Scripting

11.1.2 Python Scripting

11.2 Using Objects and Classes

11.2.1 R Scripting

11.2.2 Python Scripting

Chapter 12

Appendix 1

12.1 Comparative Syntax for Programming Constructs of R and Python

As introduced in the text, the syntax of both R and python for the basic programming constructs are reviewed there for reader convenience.

	R	Python
Basic data types and structures	scalar == vector[0] vectors, (numerical, character, logical, factor) arrays, == vectors (m x n) matrices, data frames, and lists	scalars (string, integer, float) arrays (m x n) of scalars dictionary, and lists
Operators		
conditionals		
Loops		

12.2 Extended Structures

Chapter 13

References

1. Core R Team (Ed)., R Reference Index, R Foundation for Statistical Computing, Vienna, 2018.
2. Hellmann, Doug, The Python 3 Standard Library by example, Addison-Wesley, Boston MA, 2017.
3. Python Software Foundation., Python 3.7.1 documentation, Python Software Foundation, Wilmington, Delaware, 2018
4. RStudio Consortium, RStudio Documentation, Boston, MA, 2018