Python*

UCLA Masters of Applied Economics

Fall 2018

Melody Y. Huang

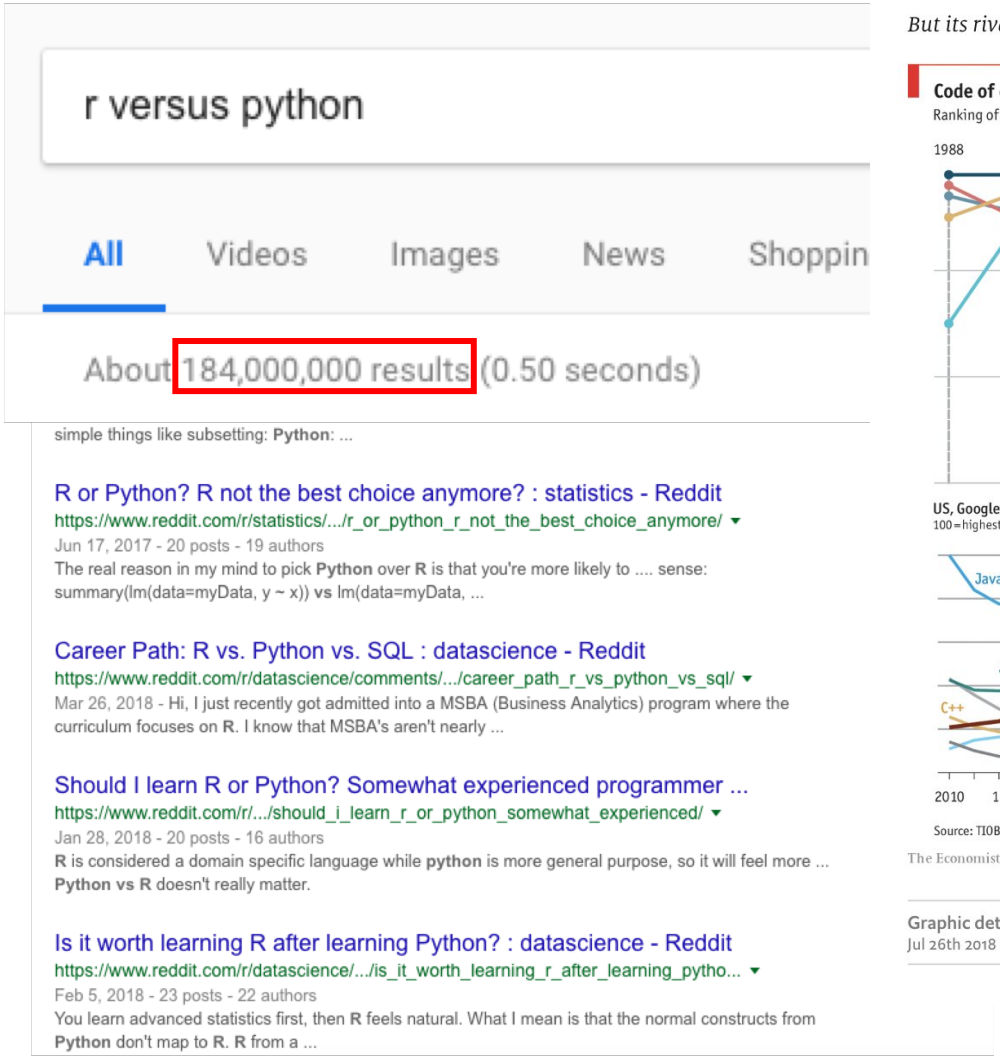
* the programming language, not the snake, in case there are any mistaken herpatologists in the room

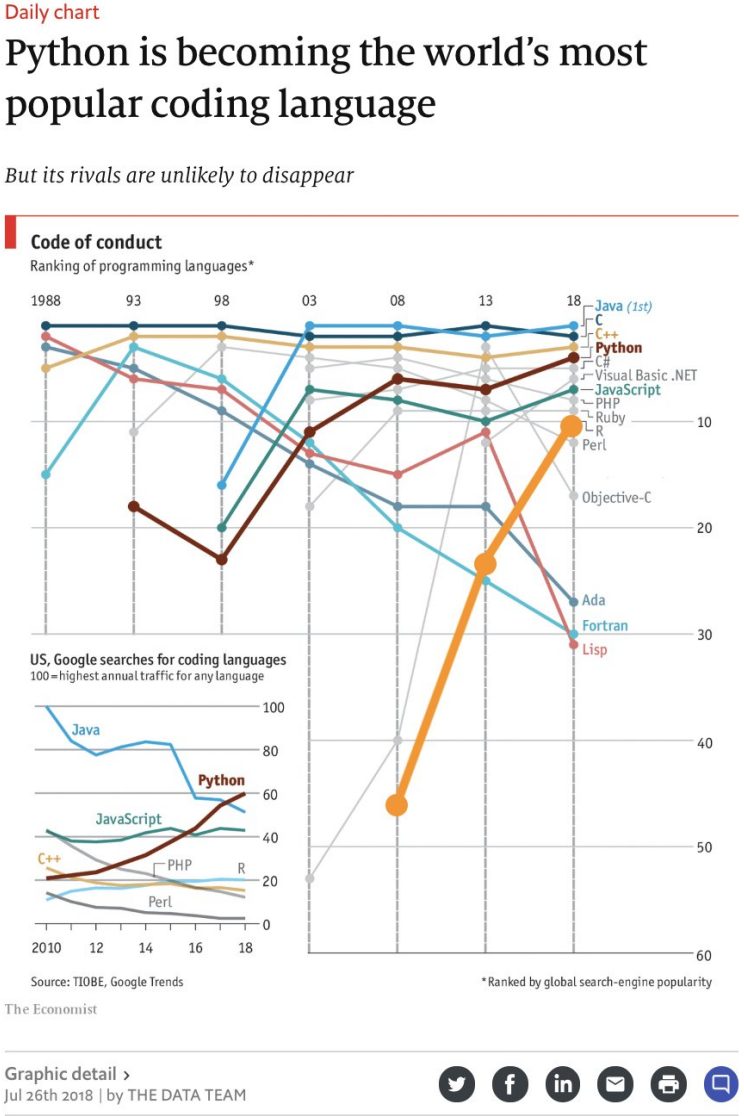
Structure of Workshop

Today:

1. Motivation
2. Background
3. Basic Syntax & Functionalities
4. Data Structures

R versus Python





R versus Python

Daily chart

Python is becoming the world's most popular coding language

But its rivals are unlikely to disappear

Code of conduct

How popular is python?

Answer Follow · 7 Request

5 Answers

Benjamin Diet, License from Claude Bernard University Lyon 1 (2018)

Answered Apr 7, 2017

Python is **fucking** popular.

Should I learn R or Python? Somewhat experienced programmer ...

https://www.reddit.com/r/.../should_i_learn_r_or_python_somewhat_experienced/

Jan 28, 2018 - 20 posts - 16 authors

R is considered a domain specific language while **python** is more general purpose, so it will feel more ... **Python vs R** doesn't really matter.

Is it worth learning R after learning Python? : datascience - Reddit

https://www.reddit.com/r/datascience/.../is_it_worth_learning_r_after_learning_pytho...

Feb 5, 2018 - 23 posts - 22 authors

You learn advanced statistics first, then **R** feels natural. What I mean is that the normal constructs from **Python** don't map to **R**. **R** from a ...

The Economist

Graphic detail ›

Jul 26th 2018 | by THE DATA TEAM

R versus Python:

- R for data analysis, statistical analysis, cleaning
- Python for machine learning, deep learning, etc.
- R has a variety of built in capabilities to quickly analyze lots of data (i.e., R's 'dataframe' objects)
  - In Python, you can do the same, but their base objects don't have the same functionalities; instead, you have to use the imported library Pandas
- Python was developed later; most of the ML functions are neatly consolidated into one package

Setting Up

- Several ways to run Python on your computer:
  - In line via computer terminal
  - As a script, via terminal
  - Anaconda
    - Spyder (Like R/R Studio type set up)
    - Jupyter (Allows you to run things line by line and have output appear in a notebook type environment)

If you plan to use a terminal-based method to run Python, I would suggest getting some sort of text editor (i.e., X-Code on Mac, or Sublime Text, which is free!)

Setting Up (cont.)

- Installing Anaconda will give you most of the packages you need

- I would encourage you to also install ‘pip’, which allows you to easily install other packages and extensions (i.e., TensorFlow) that may not come with Anaconda

White Spaces

- White spaces matter!
- There are no brackets in Python, so your computer determines the code folding by indentation/spaces

```
#In R:
for(i in 1:10){
   print(i)
}
var1<-i
print(var1)

#You can also write:
for(i in 1:10){
print(i)
} var1<-i; print(var1)
```

```
#In Python:
for i in range(10):
   print(i+1)
var1=i
print var1

for i in range(10):
   print(i+1)
   var1=i
   print(var1)
```

Indexing

- In Python, we index starting from zero
- Example:

```
x = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

  - To access the first element of x (i.e., 1): `x[0]`

Basic Functions

- `len(x)`
  - Returns the length of an object
  - Equivalent to R's `length()` function

- `set(x)`
  - Returns unique items
  - Equivalent to R's `unique()` function

- Assignment operator: = (not <-)

Range Function

- `range(x)`:
  - Returns a sequence of numbers of length x, starting from 0
  - Will create a range object – to view the contents, you have to write a for loop
  - Example:
    - `range(10)`:
      - Actually returns 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
      - NOT: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Try it for yourself!

1. Print a seq. of numbers from 0 to 100.
2. Print a seq. of even numbers from 2 to 100.

Lists

- Similar to R's numeric vectors (x <- c(1, 2, 3, 4))
- We declare lists using brackets: [1, 2, 3, 4]
- Example:

```
#Numbers:
num_list = [1,2,3,4]

#Strings
str_list = [“hello”, “world”]

#Lists
list_ception = [[1,2,3], [4,5,6], [7,8,9]]
```

Lists (cont.)

- Cool things you can do with lists:
  - `x.append()`
    - Adds stuff to the end of a list
  - `x.insert(i, x2)`
    - i = index at which we insert x2
    - Inserts element at index *i*
  - `x.pop()`
    - Takes last element from list and remove it to store it elsewhere
    - Like a stack in C++

List Comprehension

- List comprehension is a way to define lists and dictionaries in a way where you have a nested for loop within your list.
- For example, let's say I want to create a list from 1 to 10.
- You could write:

```
Numbers = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

- Alternatively, using list comprehension:

```
Numbers = [x+1 for x in range(10)]
```

List Comprehension (cont.)

- Why would we ever use this?!
- Let's say you have a really messed up list that contains lists within itself (a list-ception)
- The inner list contains 6 elements:
  1. Open Price
  2. High
  3. Low
  4. Close
  5. Volume Traded
  6. Market Cap

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
         …
        [23, 51, 23, 15, 2015, 1034951]]
```

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
         …
        [23, 51, 23, 15, 2015, 1034951]]
```

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
         …
        [23, 51, 23, 15, 2015, 1034951]]
```

This is the **first inner list** of the larger list.
So to call it, we write: `data[0]`.
But this returns the entire inner list!
We only want the first element of the inner list.

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
         …
        [23, 51, 23, 15, 2015, 1034951]]
```

- To call the first element, we treat `data[0]` as if it is any other list, and write: `data[0][0]`

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
         …
        [23, 51, 23, 15, 2015, 1034951]]
```

Solution:

```
open = [data[i][0] for i in range(len(data))]
```

List Comprehension (cont.)

- To separate each element so that you can have just a list that contains all opening prices, you can use list comprehension:

```
data = [[42, 53, 1, 25, 2300, 5112346],
        [12, 32, 52, 61, 2600, 10945209],
          …
        [23, 51, 23, 15, 2015, 1034951]]
open = [data[i][0] for i in range(len(data))]

#Equivalent to the following for loop:
open = [] #Empty list
for i in range(len(data)):
    open.append(data[i][0])
```

Dictionaries

- Also known as Python's version of a hashtable
- Each item in your dictionary has both a key and a value associated with it
- This is cool because we can input a key, and retrieve the value without iterating through the entire list!
- Defined by braces: { }
  - More specifically: {key1: value1, key1: value1, …}

Dictionaries (cont.)

Example:

- Let's say we have a dictionary containing student names and their UID. Call this, registrar.

```
registrar = {'Student1': 12345,
             'Student2': 34953,
             'Student3': 586153,
             'Student4': 20390}
```

- So, to retrieve the UID associated with Student 4, we would simply type: `registrar['Student4']`

Dictionary (cont.)

- Why use a dictionary?
  - Very computationally efficient to retrieve information!
  - We care about computational efficiency because we want to work with big data which… is big.