Introduction to Python for Text Analysis

Jennifer Pan

Institute for Quantitative Social Science
Harvard University

(Political Science Methods Workshop, February 21 2014)

*Much credit to Andy Hall and *Learning to Think Like a Computer Scientist*

Outline

Outline

How do we compare languages?

- Depends on the task
- For the same task
  - Speed of processing
  - Amount of code (how much does each line accomplish)
  - Readability of code
- There are many ways of doing the same thing in each language $\rightsquigarrow$ your preference

Comparing Python and R

- Similarities:
  - Both are high-level languages
  - Both are interpreted languages
  - Both have shell and script mode (though most rarely use script mode for R)
- General advantages of each:
  - R: more richness for data analysis, e.g., >5,000 packages for data analysis, lots of packages developed for social science
  - Python: much more varied applications, e.g., internet protocols, OS interface, data analysis

Outline

Comparing Python and R

Refer to `basics.py` for all exercises in this section:

1. Data types
2. Variables
3. Functions
4. Conditionals
5. Iteration
6. String
7. Lists
8. Modules
9. Scripting

1. Data types

Every thing you come across belongs to a **data type**:

```
>>> type(4)
<type 'int'>

>>> type(5.3226)
<type 'float'>

>>> type("Hello, World!")
<type 'str'>
>>> type('Hello, MIT')
<type 'str'>
>>> type('4')
<type 'str'>

>>> type(True)
<type 'bool'>
```

1. Data types

Data types can be converted

```
>>> int(5.3326)
5
>>> float(4)
4.0
>>> int('4')
4
>>> str(5.3326)
'5.3326'
```

2. Variables

A **variables** is a name that refers to a value. In python, a name is assigned to a value with = (equivalent of <- in R)

```
>>> message = "How are you doing?"
>>> n = 34
>>> pi = 3.14159

>>> print message
How are you doing?
>>> print n
34
>>> print pi
3.14159

>>> type(message)
<type 'str'>
>>> type(n)
<type 'int'>
```

2. Variables

Variables of the same type can be combined

```
>>> n + pi
37.14159
>>> n + message
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Variables can be altered

```
>>> count = 0
>>> count = count + 1
>>> count
1
>>> count += 1
>>> count
2
```

3. Functions

A **function** is series of statements with names that perform a desired operation. The python syntax is always (indent crucial):

```
def NAME( PARAMETER LIST ):
    STATEMENTS
```

For example:

```
>>> def funny_add(a, b):
...     return a + b + 1
...

>>> funny_add(3,4)
8
>>> weird = funny_add(3,4)
>>> weird
8
```

3. Functions

There are lots of useful built-in functions, some of which we have used

```
>>> type(pi)
<type 'float'>
>>> abs(-15)
15
>>> max(7,11)
11
>>> max(abs(-15),7,11)
15
>>> len(message)
18
```

4. Conditionals

Conditionals allows us to change our program based on pre-set conditions (e.g., scrape only entries with X). They take the form:

```
if BOOLEAN EXPRESSION:
    STATEMENTS
```

For example:

```
>>> x = 5
>>> if x > 0:
...     print "x is positive"
...
x is positive
>>> if x % 2 == 0:
...     print x, "is even"
... else:
...     print x, "is odd"
...
5 is odd
```

4. Conditionals

Conditionals can be chained:

```
>>> y = 4
>>> if x < y:
...     print x, "is less than", y
... elif x > y:
...     print x, "is greater than", y
... else:
...     print x, "and", y, "are equal"
5 is greater than 4
```

They can also be nested:

```
>>> if x == y:
...     print x, "and", y, "are equal"
... else:
...     if x < y:
...         print x, "is less than", y
...     else:
...         print x, "is greater than", y
5 is greater than 4
```

4. Conditionals

Conditionals can also be part of functions:

```
>>> def is_divisible(a, b):
...     if a % b == 0:
...         return True
...     else:
...         return False
...

>>> is_divisible(x,y)
False

>>> type(is_divisible)
<type 'function'>
```

5. Iteration

Iterators allow us to repeat execution of a set of statements (e.g., stem all of the text from each row of a csv file).

Important are the `while` statement

```
>>> def countdown(n):
...     while n > 0:
...         print n
...         n = n-1
...     print "Blastoff!"
...
>>> countdown(6)
```

And the `for` statement:

```
>>> for x in message:
...     print x
```

6. Strings

Strings are a data type that's critical for working with text data

Strings can be sliced:

```
>>> message
'How are you doing?'
>>> message[0]
'H'
>>> message[-1]
'?'
>>> message[0:5]
'How a'
>>> message[5:]
're you doing?'
```

But they are immutable:

```
>>> message[1] = "L"
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: 'str' object does not
support item assignment

>>> new_message = "L" + message[1:]
>>> new_message
'Low are you doing?'
```

6. Strings

We often use iterators to process strings:

```
>>> index = 0
>>> while index < len(message):
...     letter = message[index]
...     print letter
...     index += 1

>>> for thing in message:
...     print thing

>>> prefixes = "JKLMNOPQ"
>>> suffix = "ack"
>>>
>>> for letter in prefixes:
...     print letter + suffix
```

6. Strings

Here's an example of a function that removes vowels, using an iterator and a conditional:

```
>>> def remove_vowels(s):
...     vowels = "aeiouAEIOU"
...     s_without_vowels = ""
...     for letter in s:
...         if letter not in vowels:
...             s_without_vowels += letter
...     return s_without_vowels

>>> remove_vowels(message)
'Hw r y dng?'
```

6. Strings

There are a number of useful string methods (a method is a member of a class, here methods are functions that belong to the string class)

```
>>> message.upper()
>>> message.lower()

>>> bad_message = "     How are    you doing    ?   "
>>> bad_message.strip()
>>> bad_message.replace(" ","")
>>> bad_message.split()
>>> " ".join(bad_message.split())

>>> message.find('How')
>>> message.find('how')
>>> message.find('?')
```

6. Strings

The string formatting operator % is very useful.

```
>>> name = "John"
>>> age = 24
>>> weight = 150.4

>>> "I am %s and I am %d years old and weight %f pounds."
% (name, age, weight)
'I am John and I am 24 years old and weight 150.400000 pounds.'

>>> "I am %s and I am %d years old and weight %.1f pounds."
% (name, age, weight)
'I am John and I am 24 years old and weight 150.4 pounds.'
```

7. Lists

Lists are a very important data type because they allow you to store data for later use

```
>>> numlist = [10, 20, 30, 40]
>>> numlist
[10, 20, 30, 40]
>>> numlist[0]
10
>>> numlist[1]
20
>>> numlist[1:3]
[20, 30]
>>> numlist[3:]
[40]
```

7. Lists

Iterators are crucial for going through each item of a list

```
>>> for item in numlist:
...     print item
```

You can create a numerical list with the `range` function:

```
>>> range(1,5)
[1, 2, 3, 4]
>>> range(5)
[0, 1, 2, 3, 4]
```

7. Lists

You can create any list with the `.append()` method:

```
>>> democracies = []
>>> democracies
[]
>>> democracies.append("United States")
>>> democracies.append("Germany")
>>> democracies.insert(0,"Canada")
>>> democracies
['Canada', 'United States', 'Germany']
```

Lists are mutable:

```
>>> democracies[0]
'Canada'
>>> democracies[0] = "India"
>>> democracies
['India', 'United States', 'Germany']
```

8. Modules

A module is a neatly packaged set of functions. Some come with python, others need to be installed. You need to `import` modules to use them (like library() in R)

Some useful modules:

- `import os`: allows you to interact with your operating system, e.g., set working directory, set the path to file
- `import time`: time functions (e.g., for benchmarking)
- `import string`: lots of string operations, some similar to the methods available to string data type, others as well, e.g.,

```
>>> string.find(message, 'are')
>>> string.split(message)

>>> string.printable
>>> string.digits
```

8. Modules

Two modules for web-scraping we'll use are
`urllib2` and `BeautifulSoup`

```
>>> import urllib2
>>> url = "http://en.wikipedia.org/wiki/Adult_lifetime_cannabis_use_by_
>>> page = urllib2.urlopen(url)
>>> type(page)
>>> page = urllib2.urlopen(url).read()
>>> type(page)
>>> print page

>>> from bs4 import BeautifulSoup
>>> soup = BeautifulSoup(page)
>>> print soup.prettify()
```

9. Scripting

We've been working in the command interface, but if you're doing anything that requires more than a few lines of code (i.e., web-scraping, text analysis), you don't want to input each line of code manually, so you need a **script**. A script is just lines of code for python to execute, that you save as a file ending in `.py`.

`cleanup.py` is a script that cleans up text

Run the script by cd to the directory where the file is, and type:

```
$ python cleanup.py
I cleaned the bad string:'    How   are you   doing  ?    '
made it into a good string: 'How are you doing ?'
and it took 0.000017 seconds
```

`cleanup.py` has about 10 lines of executable code. How many lines to do this in R? Can you do it in R without RegEx?

Outline

Web-scraping example

We're going to do the same thing Danny did in the last workshop

Code found in `scraping.py`, does the following things

- Load needed packages
- Define the `url` we're scraping from
- Use `BeautifulSoup` to find the country name and rate
- Save country name and rate as items of a list
- Fix a problem with Northern Ireland (because of bad HTML)
- Save output of country names and rates to .csv

Web-scraping example stats

Here are the stats for this code:

- 25 lines of code (not including comments)
- Does everything in 15-20 seconds

How does this compare to R?

- Amount of code
- Readability of code
- Processing speed

Outline

Comparing Available Resources

	Python	R
Access URL	`urllib2`	`ReadLines`

Clean HTML	`BeautifulSoup`	Regular expressions
		`grep`
		`gsub`

Pre-processing	`nltk`	`tm`
Classification	`nltk` or `scikit-learn`	`tm`
Topic models	`gensim`	`topicmodels`
Topic models	More info	More info

- Pre-processing include: segmenting, tokenizing, stripping, stopword removal, stemming, creating a TDM
- It's very easy to write wrappers for other languages in python (e.g., Stanford NLP group has lots of great tools in Java, it's easy to access these tools with python), can be harder with R.

For more information

bit.ly/JenPan_Python

Please send corrections to jjpan@fas.harvard.edu