Text Mining
Intro to Python

Mattias Villani, Johan Falkenjack

NLPLAB
Dept. of Computer and Information Science
Linköping University

Overview

- What is Python? How is it special?
- Python's objects
- If-else, loops and list comprehensions
- Functions
- Classes
- Modules

What is Python?

- First version in 1991
- **High-level language**
- Emphasizes **readability**
- **Interpreted** (bytecode .py and .pyc) [can be compiled via C/Java]
- Automatic memory management
- Strongly dynamically typed
- **Functional and/or object-oriented**
- **Glue** to other programs (interface to C/C++ or Java etc)
- Popular in data science ("Prototype in R, implement in Python")
- Two currently developed versions, 2.x and 3.x
    - This course uses Python 2.7

The Benevolent(?) Dictator For Life (BDFL) Guido van Rossum

Python peculiarites (compared to R/Matlab)

- Not primarily a numerical language.
- **Indexing begins at 0**, as indexes refer to breakpoints between elements.
- It follows that `myVector[0:2]` returns the first and second element, but not the third.
- **Integer division** by default in 2.x. `from __future__ import division`.
- **Indentation matters!**
- Can import specific functions from a module.
- Assignment **by object**, **NOT by copy** or **by reference**.
    - Approximately, assignment **by copy of reference**.
- `a = b = 1` assigns 1 to both `a` and `b`.

PYTHON'S OBJECTS

- Built-in types: **numbers**, **strings**, **lists**, **dictionaries**, **tuples** and **files**.
- **Vectors**, **arrays** and **matrices** are available in the **numpy/scipy** modules.
- Python is a **strongly typed** language. `'johan' + 3` gives an error.
- Python is a **dynamically typed** language. No need to declare a variables type before it is used. Python figures out the object's type.
- Implication: Polymorphism by default:
  - "In other words, don't check whether it IS-a duck: check whether it QUACKS-like-a duck, WALKS-like-a duck, etc, etc, depending on exactly what subset of duck-like behaviour you need to play your language-games with." - Alex Martelli

STRINGS

- `s = 'Spam'`
- `s[0]` returns first letter, `s[-2]` return next to last letter. `s[0:2]` returns first **two** letter.
- `len(s)` returns the number of characters.
- `s.lower()`, `s.upper()`, `s.count('m')`, `s.endswith('am')`, `...`
- **Which methods are available for my object?** Try in Spyder: type `s.` followed by TAB.
- + operator **concatenates strings**.
- (behind the scenes: the string object has an `__add__` method: `s.__add__(anotherString)`)
- `sentence = 'Guido is the benevolent dictator for life'.sentence.split()`
- `s*3` returns `'SpamSpamSpam'`supported and

The list object

- A list is a **container of several variables**, possibly of different types.
- `myList = ['spam','spam','bacon',2]`
- The list object has several associated **methods**
  - `myList.append('egg')`
  - `myList.count('spam')`
  - `myList.sort()`
- + operator concatenates lists: `myList + myOtherList` merges the two lists as one list.

The list object

- Extract elements from a list: `myList[1]`
- Lists inside lists:
  - `myOtherList = ['monty','Python']`
  - `myList[1] = myOtherList`
  - `myList[1]` returns the list `['monty','Python']`
  - `myList[1][1]` returns the string 'Python'

Strings again

- Strings are **immutable**, i.e. can't be changed after creation.
- Every “change” creates a new string.
- Try to avoid creating more strings than necessary:
  - Avoid: `my_string = 'Python ' + 'is ' + 'fun!'`
  - Instead: `' '.join(['Python', 'is', 'fun'])`
- In loops where you construct strings, add constituents to a list and join after loop finishes.

Tuples

- `myTuple = (3,4,'johan')`
- **Like lists, but immutable**
- Why?
  - Faster than lists
  - Protected from change
  - Can be used as keys in dictionaries
  - Multiple return object from function
  - Swapping variable content `(a, b) = (b, a)` (`[a,b = b,a]` also works)
  - String formatting: `name = "Johan"; age = 30; "My name is %s and I am %d years old" % (name , age)`
  - Sequence unpacking `a , b, c = myTuple`
- `list(myTuple)` returns myTuple as a list. `tuple(myList)` does the opposite.

Vectors and arrays (and matrices)

- `from scipy import *`
- `x = array([1,7,3])`
- 2-dimensional **array** (matrix): `X = array([[2,3],[4,5]])`
- **Indexing arrays**
  - First row: `X[0,]`
  - Second column: `X[,1]`
  - Element in position 1,2: `X[0,1]`
- Array **multiplication** (*) is element-wise, for matrix multiplication use `dot()`.
- There is also a **matrix object**: `X = matrix([[2,3],[4,5]])`
  - **Arrays are preferred** (not matrices).
- Submodule **scipy.linalg** contains a lot of **matrix-functions** applicable to arrays (`det()`, `inv()`, `eig()` etc). I recommend: `from scipy.linalg import *`

DICTIONARIES

- **Unordered** collection of objects (elements).
- `myDict = {'Sarah':29, 'Erik':28, 'Evelina':26}`
- Elements are **accessed by keyword not by index** (offset): `myDict['Evelina']` returns 26.
- **Values can contain any object**: `myDict = {'Marcus':[23,14], 'Cassandra':17, 'Per':[12,29]}`. `myDict['Marcus'][1]` returns 14.
- Any immutable object can be a key: `myDict = {2:'contents of box2', (3, 'a'):'content of box 4', 'blackbox':10}`
- `myDict.keys()`
- `myDict.values()`
- `myDict.items()`

Sets

- **Set**. Contains objects in **no order** with **no identification**.
  - With a **sequence**, elements are ordered and identified by position. `myVector[2]`
  - With a **dictionary**, elements are unordered but identified by some key. `myDict['myKey']`
  - With a **set**, elements stand for themselves. No indexing, no key-reference.
- Declaration: `fib=set( (1,1,2,3,5,8,13) )` returns the set `([1, 2, 3, 5, 8, 13])`
- Supported methods: `len(s)`, `x in s`, `set1 < set2`, `union`, `intersection`, `add`, `remove`, `pop ...`

Boolean operators

- `True/False`
- `and`
- `or`
- `not`
- `a = True; b = False; a and b` [returns False].

IF-ELSE CONSTRUCTS

IF-ELSE STATEMENT

```
a =1
if a==1:
    print('a is one')
elif a==2:
    print('a is two)
else:
    print('a is not one or two')
```

- **Switch statements** via dictionaries (see Jackson's Python book).

While loops

while loop

```
a =10
while a>1:
   print('bigger than one')
   a = a - 1
else:
   print('smaller than one')
```

For loops

- **for loops can iterate over any iterable.**
- **iterables**: strings, lists, tuples

FOR LOOP

```
word = 'mattias'
for letter in word:
  print(letter)
myList = ['']*10
for i in range(10):
  myList = 'mattias' + str(i)
```

List comprehensions

- As in R, loops can be slow. For small loops executed many times, try list comprehensions:
- Set definition in mathematics

$$\{x \text{ for } x \in \mathcal{X}\}$$

  where $\mathcal{X}$ is some a finite set.

$$\{f(x) \text{ for } x \in \mathcal{X}\}$$

- List comprehension in Python:
  - `myList = [x for x in range(10)]`
  - `myList = [sin(x) for x in range(10)]` (don't forget `from math import sin`)
  - `myList = [x + y for x in linspace(0.1,1,10) for y in linspace(10,100,10)]` (`from scipy import linspace`)

Defining functions and classes

Defining functions

```
def mySquare(x):
    return x**2
```

- Calling the function: `mySquare(x)`
- Instance functions in classes are defined similarly using the `self` reference.
- Make you own module by putting several functions in a .py file. Then import what you need.

Misc

- Comments on individual lines starts with #
- Doc-strings can be used as comments spanning over multiple lines but this should be avoided `"""This is a looooong comment"""`