Text Mining
Intro to Python

Mattias Villani

Statistics
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)

The Benevolent Dictator For Life (BDFL) Giudo van Rossum

Python peculiarites (compared to R/Matlab)

- **Counting begins at 0**.
- myVector[0:2] returns the first and second element, but not the third.
- $3/2 = 1$. **Integer division**. `from __future__ import division`.
- **Indentation matters!**
- Can import specific functions from a module.
- Some variable **assignments** are **by reference**, others are **by copy**.
- `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. 'mattias' + 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.

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 letters.
- `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'`

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'

Python's objects: vectors and arrays (and matrices)

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

Python's objects: dictionaries

- **Unordered** collection of objects (elements).
- `myDict = {'Leif':23, 'Dag':17, 'Lyam':12}`
- Elements are **accessed by keyword not by index** (offset): `myDict['Dag']` returns 17.
- **Can contain any object**: `myDict = {'Leif':[23,14], 'Dag':17, 'Lyam':[12,29]}`. `myDict['Leif'][1]` returns 14.
- Numbers can also be used as keys: `myDict = {2:'contents of box2', 4:'content of box 4', 'blackbox':10}`
- `myDict.keys()`
- `myDict.values()`
- `myDict.items()`

Python's objects: tuples

- `myTuple = (3,4,'mattias')`
- **Like lists, but immutable** (cannot change elements are creation)
- 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 = "Mattias"; age = 39; "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.

Python's objects: 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 one')
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')
```

Loops

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

FOR LOOP

```
word = 'mattias'
for letter in word:
   print(letter)
```

LOOPS, CONT.

FOR LOOP 2

```
myList = [’’]*10
for i in range(10):
   myList = ’mattias’ + str(i)
```

FOR LOOP 2

```
myList = [’’]*10
for i in range(10):
   myList = ’mattias’ + str(i)
```

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)`
- Classes are defined similarly using the self object.
- Make you own module by putting several functions in a .py file. Then import what you need.

Misc

- Comments one individual lines starts with #
- Comments spanning over multiple lines """This is a looooong comment''''''