# Python for Finance - an Introduction

José Pedro Silva

December 7, 2014

# Contents

## 0.1 Workshop Python for Finance

## 0.2 - an Introduction

*José Pedro Silva* 1 - silva@math.uni-wuppertal.de

#### Date

9-11 Dec 2014

#### Location:

ISEG Rua do Quelhas, 6 1200-781 Lisboa, PORTUGAL

### 0.2.1 Brief description

This workshop is aimed at users with few or no proficiency in Python who want to apply numerical methods in Finance. A short introduction to the language and its syntax will be given, as well as a brief description of the most important modules (NumPy, SciPy, Matplotlib, Pandas, IPython) for scientific and financial analysis purposes. The workshop will consist of interactive lectures with several examples followed by practical sessions.

- ### 0.2.2 Day1
  - Python
  - IPython
  - Numpy
  - Matplotlib

### 0.2.3 Day2

- - SciPy
  - Pandas
  - Tips and Tricks
  - Benchmarks
  - Randomness
  - FFT
  - Option Pricing

# Chapter 1

# More information

- For Python Quants Conference
- Quantopian
- ContinuumIO Python for Finance course

In[1]:
```
%load_ext version_information
%version_information numpy, matplotlib, scipy, pandas
```

Out[1]:

| Software | Version |
|---|---|
| Python | 2.7.8 —Anaconda 2.1.0 (64-bit)— (default, Aug 21 2014, 18:22:21) [GCC 4.4.7 20120313 (Red Hat 4.4.7-1) |
| IPython | 2.3.1 |
| OS | posix [linux2] |
| numpy | 1.9.1 |
| matplotlib | 1.4.2 |
| scipy | 0.14.0 |
| pandas | 0.15.1 |
| Fri Dec 05 10:35:46 2014 CET | |

*All notebooks will be available from the official ITN-Strike repository soon*

In[2]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[2]:
```
<IPython.core.display.HTML at 0x7f2fa3d8abd0>
```

In[]:
```
```

## 1.1 I.II-Iterables

An iterable is a python container with two special methods: **iter** and **next**. There are some built-in iterables like lists, tuples, dicts, strings, files, etc...

As the name hints, an iterable allows iteration over its items, usually with a for statement:

```
for item in container:
     print item    #or do sth else with item
```

Iterating is the process of accessing and returning each item from a container

In [4]:
```
my_list = [1,2,3]
for i in my_list:
    print i
```

**List**

```
1
2
3
```

In [5]:
```
'__iter__' in dir(my_list)
```

Out[5]:
```
True
```

In [4]:
```
my_list = [j for j in range(5)]
for i in my_list:
    print i
```

```
0
1
2
3
4
```

In [9]:
```
a = (1,2,3)
a
```

**Tuples** Out[9]:
```
(1, 2, 3)
```

In [11]:
```
for item in a:
    print item
```

```
1
2
3
```

In [13]:
```
from numpy.random import randn
a = dict(zip(range(3),randn(3)))
```

In[15]:
```
for item in a:
    print item
```

## Dicts

```
0
1
2
```

In[18]:
```
for item in a.iterkeys():
    print item
```

```
0
1
2
```

In[19]:
```
for item in a.itervalues():
    print item
```

```
-0.445128793029
0.316719007573
2.06412261427
```

In[21]:
```
for item in a.iteritems():
    print item
```

```
(0, -0.44512879302937886)
(1, 0.31671900757330107)
(2, 2.0641226142684475)
```

In[20]:
```
for k,v in a.iteritems():
    print k,v
```

```
0 -0.445128793029
1 0.316719007573
2 2.06412261427
```

In[24]:
```
a = 'Python is awesome!'
```

In[25]:
```
for item in a:
    print item
```

## Strings

```
P
y
t
h
o
n

i
s

a
w
e
s
o
m
e
!
```

In[38]:
```
f = open('iter_text.txt','rb')
```

In[39]:
```
for item in f:
    print item
```

**Files**

```
Line One

Line Two

Last line
```

In[44]:
```
for item in f.readlines():
    print item
```

In[45]:
```
f = open('iter_text.txt','rb')
for item in f.readlines():
    print item
```

```
Line One

Line Two

Last line
```

We have two for each line, one from the text and one from the print statement. A quick trick surpresses the one from print

In[46]:
```
f = open('iter_text.txt','rb')
for item in f:
    print item,
```

```
Line One
Line Two
Last line
```

**Note**: As long as possible, use *with* (controlled execution) which closes the file for you. So instead of

```
f = open(filename,'r')
do something with f
f.close()
```

you can write

```
with open(filename,'r') as f:
    do something with f
```

where it automatically takes care of the clean-up for you

## 1.2 Generators

Generators are iterators which can only be iterated once and generate the values on the fly

In[52]:
```
my_generator = (j for j in range(5))
for i in my_generator:
    print i
```

```
0
1
2
3
4
```

In[55]:
```
my_generator.next()
```

```
    ---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)

    <ipython-input-55-125f388bb61b> in <module>()
----> 1 my_generator.next()


    StopIteration:
```

### 1.2.1 Yield

*yield* is the *return* of generators. It outputs one value as the *next()* method of the generator is called

In[6]:
```
def createGenerator():
    mylist = range(5)
    for i in mylist:
        yield i
```

In[7]:
```
mygenerator = createGenerator()
```

In[8]:
```
print(mygenerator)
```

```
<generator object createGenerator at 0x7f2a5474fe60>
```

In[9]:
```
for i in mygenerator:
    print i
```

```
0
1
2
3
4
```

In[10]:
```
class Bank():
    crisis = False
    def create_atm(self):
        while not self.crisis:
            yield '100'
```

In[11]:
```
BES = Bank()
```

In[12]:
```
Atm_Rossio = BES.create_atm()
```

In[14]:
```
print(Atm_Rossio.next())
```

```
100
```

In[15]:
```
print(Atm_Rossio.next())
```

```
100
```

In[16]:
```
BES.crisis = True
```

In[18]:
```
Atm_Rossio.next()
```

```
    ---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)

    <ipython-input-18-658b8cbaec0d> in <module>()
----> 1 Atm_Rossio.next()


    StopIteration:
```

In[19]:
```
BES.crisis = False
```

In[20]:
```
print(Atm_Rossio.next())
```

```
    ---------------------------------------------------------------------------
StopIteration                             Traceback (most recent call last)

    <ipython-input-20-6b8e2bf0f31f> in <module>()
----> 1 print(Atm_Rossio.next())


    StopIteration:
```

In[21]:
```
Atm_NovoBanco = BES.create_atm()
print(Atm_NovoBanco.next())
```

```
100
```

In[3]:
```
from IPython.core.display import HTML
def css_styling():
    styles = open("./styles/custom.css", "r").read()
    return HTML(styles)
css_styling()
```

Out[3]:
```
<IPython.core.display.HTML at 0x7fa1d82598d0>
```

In[]:
```

```