

# Python for Finance

## Introduction and Basics of Python

**Andras Niedermayer**



# Outline

- 1 Introduction
- 2 Why Python?
- 3 Python installation and environments
- 4 First Steps in Python
- 5 Variables
- 6 Basic Operations and Modules
- 7 Antigravity

# About Myself

- Andras Niedermayer
- Professor of Economics at Université Paris-Dauphine
- Co-founder and CTO of a fintech company (SolvencyAnalytics AG) that uses Python for data analysis
- mail: [andras.niedermayer@dauphine.fr](mailto:andras.niedermayer@dauphine.fr)
- office: P135
- slides will be available at <http://andras.niedermayer.ch>

# Literature

## Textbook

Hilpisch, Yves, *Python for Finance: Analyze Big Financial Data*, 2015, O'Reilly Publishing

## Textbook

Yan, Yuxing, *Python for Finance: Build real-life Python application for quantitative finance and financial engineering*, 2014, Pack Publishing

## Other sources

- Python 2.7 documentation at: <https://docs.python.org/2/>.
- Stanford lectures at <http://web.stanford.edu/~arbenson/cme193.html>
- UPenn lectures at <http://www.cis.upenn.edu/~cis192/spring2014/>  
Forums such as <http://stackoverflow.com/>

# Structure of Course

- Introduction to the Python Programming Language
  - ① Basics of Python
  - ② Data structures and first application
  - ③ Random numbers and Monte Carlo
  - ④ Strings, files and internet access
  - ⑤ Playing with data structures
  - ⑥ Automating tasks
- Financial Applications with Python
  - ① Learning to use the Spyder Integrated Development Environment and learning basic features of the Python programming language
  - ② Data analysis with the pandas library
  - ③ Regressions, Interpolations, and Optimization
  - ④ Simulations and Option Pricing
  - ⑤ Portfolio management and principal component analysis
  - ⑥ GARCH models, measures of market risk (Value at Risk and Expected Shortfall)

# Outline

- ① Introduction
- ② Why Python?
- ③ Python installation and environments
- ④ First Steps in Python
- ⑤ Variables
- ⑥ Basic Operations and Modules
- ⑦ Antigravity

# Why Python?

- ① *Very Popular, Most Popular Language for (Big) Data Analysis both in Academia and Industry:* see e.g. <http://pyp1.github.io/PYPL.html>
- ② *Interactive data analytics:* immediate feedback and intuitive data processing in IPython
- ③ *Powerful Libraries:* see e.g. <https://plot.ly/ipython-notebooks/markowitz-portfolio-optimization/>, <https://xkcd.com/353/>
- ④ *Open source:* free, powerful, and a very dynamic community to develop new packages.
- ⑤ *Cross-platform:* for Windows, OS X, and Linux.
- ⑥ *Intuitive language:* Closest to mathematical language and pseudocode.
- ⑦ *Dynamically typed:* One does not need to declare variable types statically (some do not like this!)

# Outline

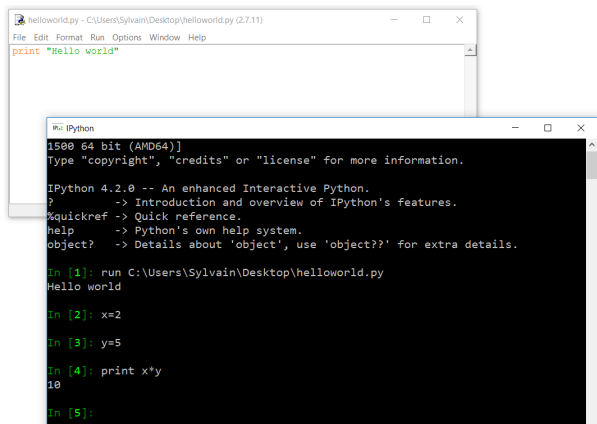
- 1 Introduction
- 2 Why Python?
- 3 Python installation and environments**
- 4 First Steps in Python
- 5 Variables
- 6 Basic Operations and Modules
- 7 Antigravity



# Anaconda

- ① One of the two major distributions of Python, along with Canopy (Enthought).
- ② Download (free! and cross-platform) from <https://store.continuum.io/cshop/anaconda/>
- ③ Includes all the important packages for this course (any many others).
- ④ There are **two** versions of Python: 2.7.x and 3.x. Syntax is slightly different, functionality almost identical. In this course we focus on 2.7.x.
- ⑤ Two (recommended) IDEs: IPython shell and Spyder – I use the first.

# IPython (cross-platform)



The image shows two overlapping windows. The top window is a text editor titled 'helloworld.py - C:\Users\Sylvain\Desktop\helloworld.py (2.7.11)'. It contains a single line of Python code: `print "Hello world"`. The bottom window is an IPython terminal titled 'IPython'. It displays the following text:

```
1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
?                -> Introduction and overview of IPython's features.
%quickref        -> Quick reference.
help             -> Python's own help system.
object?         -> Details about 'object', use 'object??' for extra details.

In [1]: run C:\Users\Sylvain\Desktop\helloworld.py
Hello world

In [2]: x=2

In [3]: y=5

In [4]: print x*y
10

In [5]:
```

# Jupyter Notebook (cross-platform, browser based)

The image displays two overlapping Jupyter Notebook windows. The foreground window, titled "Lorenz Differential Equations (autosaved)", shows a notebook with the following content:

**Exploring the Lorenz System**

In this Notebook we explore the [Lorenz system](#) of differential equations:

$$\begin{aligned}\dot{x} &= \sigma(y - x) \\ \dot{y} &= \rho x - y - xz \\ \dot{z} &= -\beta z + xy\end{aligned}$$

This is one of the classic systems in non-linear differential equations. It exhibits a range of complex behaviors as the parameters  $(\sigma, \beta, \rho)$  are varied, including what are known as chaotic solutions. The system was originally developed as a simplified mathematical model for atmospheric convection in 1963.

```
In [7]: Interact(Lorenz, N=Fixed(10), angle=(0.,360.),
                sigma=(0.0,50.0),beta=(0.,5), rho=(0.0,50.0))
```

angle: 308.2  
max\_time: 12  
 $\sigma$ : 10  
 $\beta$ : 2.6  
 $\rho$ : 28

The background window shows a "Welcome to Jupyter" page with a warning message: "WARNING: Don't rely on this server. Your server is hosted that..." and instructions on how to run Python code.

# Spyder (cross-platform)

The screenshot displays the Spyder Python IDE interface. The main window is titled "Spyder (Python 2.7)". The menu bar includes "Fichier", "Édition", "Recherche", "Source", "Exécution", "Débugger", "Consoles", "Outils", "Affichage", and "Aide". The toolbar contains icons for file operations, execution, and debugging. The main editor shows a Python script named "temp.py" with the following code:

```
1 import numpy as np
2
3 variable = np.log(2)
4
5 print "Natural logarithm of two is", variable
6
```

The "Explorateur de variables" (Variable Explorer) panel on the right shows a table with the following data:

Nom	Type	Taille	Valeur
variable	float64	1	0.69314718055994529

The "Console Python" panel at the bottom shows the output of the script:

```
Python 2.7.11 [Anaconda 4.0.0 (64-bit)] (default, Feb 16 2016, 09:58:36) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license()" for more information.

IPython 4.2.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
?quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.
?gui? -> A brief reference about the graphical user interface.

In [1]: runfile('G:/Dauphine-SDFI/Annees_16-17/PythonFinance_M1/MyCodes_PythonFinance/Codes_PythonFinance_L1/temp.py',
wdir='G:/Dauphine-SDFI/Annees_16-17/PythonFinance_M1/MyCodes_PythonFinance/Codes_PythonFinance_L1')
Natural logarithm of two is 0.69314718056

In [2]:
```

# Outline

- ① Introduction
- ② Why Python?
- ③ Python installation and environments
- ④ First Steps in Python**
- ⑤ Variables
- ⑥ Basic Operations and Modules
- ⑦ Antigravity

# Launching Python

- Open a terminal (`<Windows>+cmd`)
- type `ipython` to launch the interactive Python interpreter
- alternative: type `jupyter notebook`, in the browser window that opens, select `New` → `Notebook: Python 2`

# Arithmetic operators

In the interpreter, write:

```
In [1]: 1 + 1
Out [1]: 2
In [2]: a = 1
In [3]: print a
1
In [4]: a + 2
Out [4]: 3
```

# Quitters

```
In [1]: quit()
```



# First Usage of Python

Assume we have a return of 100 € in a few years and an interest rate of 10%. The present value of this payment is then

$$PV = \frac{FV}{(1 + R)^n}$$

where  $PV$  is the present value,  $FV$  the future value,  $R$  is the interest rate and  $n$  the number of years

What is the value of a payment in one year?

```
In [1]: 100 / (1+0.1)
Out [1]: 90.909090909090909090909090
```

# First Usage of Python

What about a payment in two years?

```
In [1]: 100 / (1+0.1)^2
-----
-----
TypeError
Traceback (most recent call last)
<ipython-input-1-6e22829b2b81> in <module>()
----> 1 100/(1+0.1)^2

TypeError: unsupported operand type(s) for ^:
'float' and 'int'
```

In Python you have to write “\*\*” instead of “^” for power

```
In [1]: 100 / (1+0.1)**2
Out [1]: 82.64462809917354
```

## Pay attention to upper and lower case

```
In [1]: x=2
In [2]: X
-----
-----
NameError
Traceback (most recent call last)
<ipython-input-4-253bcac7dd80> in <module>()
----> 1 X

NameError: name 'X' is not defined
```

Python is case sensitive.

# Dynamically Typed Variables

In Python, variables are dynamically typed, so that they can change their type at runtime

```
In [1]: x=2  
In [2]: x  
In [3]: x='aa'  
In [4]: x
```

Python is case sensitive.

# Help

Help!

```
In [1]: help()
```

- Google + Stackoverflow
- <http://python.org>

# Exercises

- Calculate the area of a circle with a radius of 10 cm
- Calculate the diagonal of a square with side length 1
- What is the circumference of a trapezoid with side lengths of 127 m each?

# Solutions

- Exercise 1

```
In [1]: area = 3.14159 * 10**2  
In [2]: area  
Out [2]: 314.159
```

- Exercise 2

```
In [1]: diag = 2**0.5  
In [2]: diag  
Out [2]: 1.4142135623730951
```

- Exercise 3

```
In [1]: 127 + 127 + 127 + 127  
Out [1]: 508
```

# Outline

- 1 Introduction
- 2 Why Python?
- 3 Python installation and environments
- 4 First Steps in Python
- 5 Variables**
- 6 Basic Operations and Modules
- 7 Antigravity



# Variables

Let's assign values to variables

```
In [1]: a = 1
In [2]: aa = 'hello'
In [3]: b = 2
In [4]: a + b
Out [4]: 3
```

Print the values of variables

```
In [1]: a
Out [1]: 1
In [2]: aa
Out [2]: 'hello'
In [3]: print a
1
In [4]: print aa
'hello'
```

# Variables

Variables change values!

```
In [1]: x = 5
In [2]: x
Out [2]: 5
In [3]: x = x + 1
In [4]: x
Out [4]: 6
```

Variables in programming languages are different than variables in mathematics

# Variables

Some errors:

```
In [1]: aaa
-----
NameError
Traceback (most recent call last)
<ipython-input-20-5c9597f3c824> in <module>()
----> 1 aaa
NameError: name 'aaa' is not defined
```

```
In [1]: sqrt(2)
-----
NameError
Traceback (most recent call last)
<ipython-input-21-40e415486bd6> in <module>()
----> 1 sqrt(2)
NameError: name 'sqrt' is not defined
```

# Variables

Bad naming of variables:

```
In [1]: x = 100
In [2]: y = 0.1
In [3]: z = x / (1 + y)
In [4]: print "The result is ", z
The result is 90.90909090
```

Good naming of variables

```
In [1]: FV = 100
In [2]: R = 0.1
In [3]: PV = FV / (1 + R)
In [4]: print "The result is ", PV
The result is 90.90909090
```

# Variables

Introspection:

```
In [1]: dir()
Out [1]: ['__builtin__', '__doc__',
          '__name__', '__package__']
```

dir() allows you to access all define variable/names

```
In [1]: a = 1
In [2]: dir()
Out [2]: ['__builtin__', '__doc__',
          '__name__', '__package__', 'a']
```

# Variables

Deleting a variable:

```
In [1]: a = 1
In [2]: dir()
Out [2]: ['__builtin__', '__doc__',
'__name__', '__package__', 'a']
In [3]: del a
In [4]: dir()
Out [4]: ['__builtin__', '__doc__',
'__name__', '__package__']
```

# Outline

- ① Introduction
- ② Why Python?
- ③ Python installation and environments
- ④ First Steps in Python
- ⑤ Variables
- ⑥ Basic Operations and Modules**
- ⑦ Antigravity

# Basic Operations

Basic Operations:

```
In [1]: 1+1
In [2]: 1-2
In [3]: 2*2
In [4]: 3/2
In [5]: 3/2.
In [6]: 3//2.
In [7]: int(2.5)
```



# The Mathematics module

For mathematics, use the module:

```
In [1]: dir()
In [2]: import math
In [3]: dir()
In [4]: dir(math)
In [5]: math.sqrt(2)
In [6]: math.pow(2,2)
```

# Help Again

For help for a specific function or module:

```
In [1]: help(math.pow)
In [2]: help(math)
```

# Outline

- ① Introduction
- ② Why Python?
- ③ Python installation and environments
- ④ First Steps in Python
- ⑤ Variables
- ⑥ Basic Operations and Modules
- ⑦ Antigravity**

# Antigravity

```
In [1]: import antigravity
```