Python for Finance

Introduction and Basics of Python

Andras Niedermayer

DAUPHINE
UNIVERSITÉ PARIS

Outline

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
- office: P135
- slides will be availabe 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 anlasysis
  6. GARCH models, measures of market risk (Value at Risk and Expected Shortfall)

Outline

Why Python?

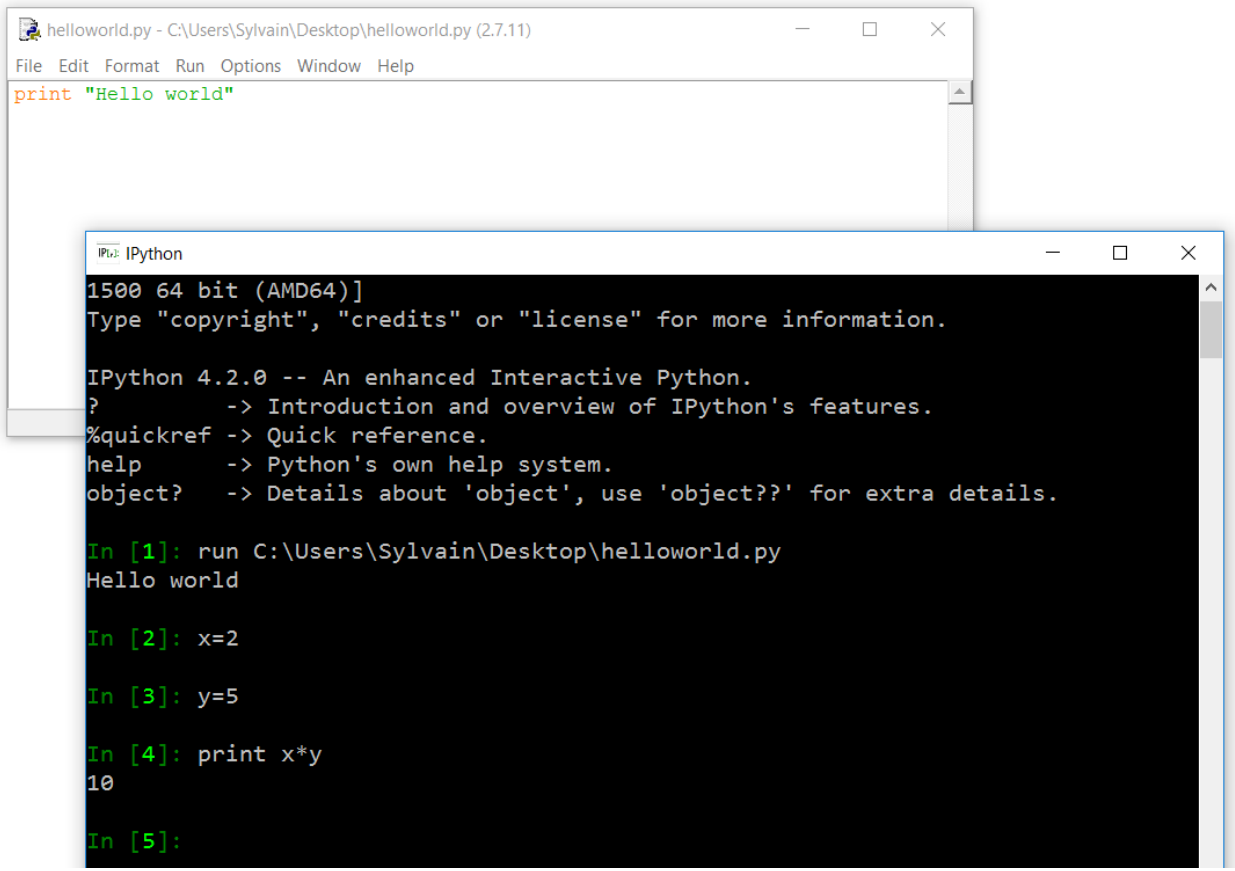
1. *Very Popular, Most Popular Language for (Big) Data Analysis both in Academia and Industry:* see e.g. `http://pypl.github.io/PYPL.html`
2. *Interactive data analytics:* immediate feedback and intuitive data processing in `IPython`
3. *Powerful Libraries:* see e.g. `https://plot.ly/ipython-notebooks/markowitz-portfolio-optimization/`, `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

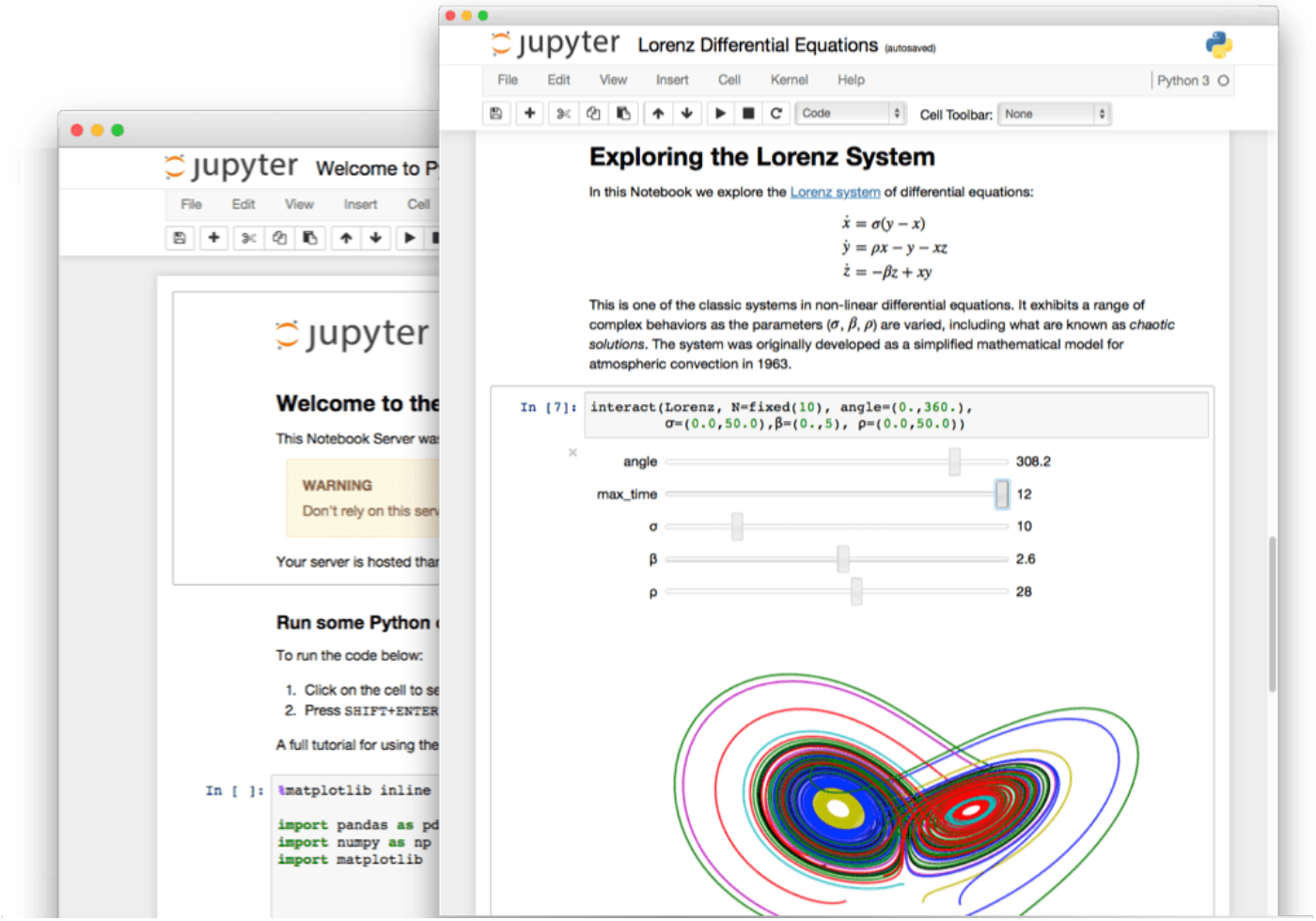
Anaconda

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.

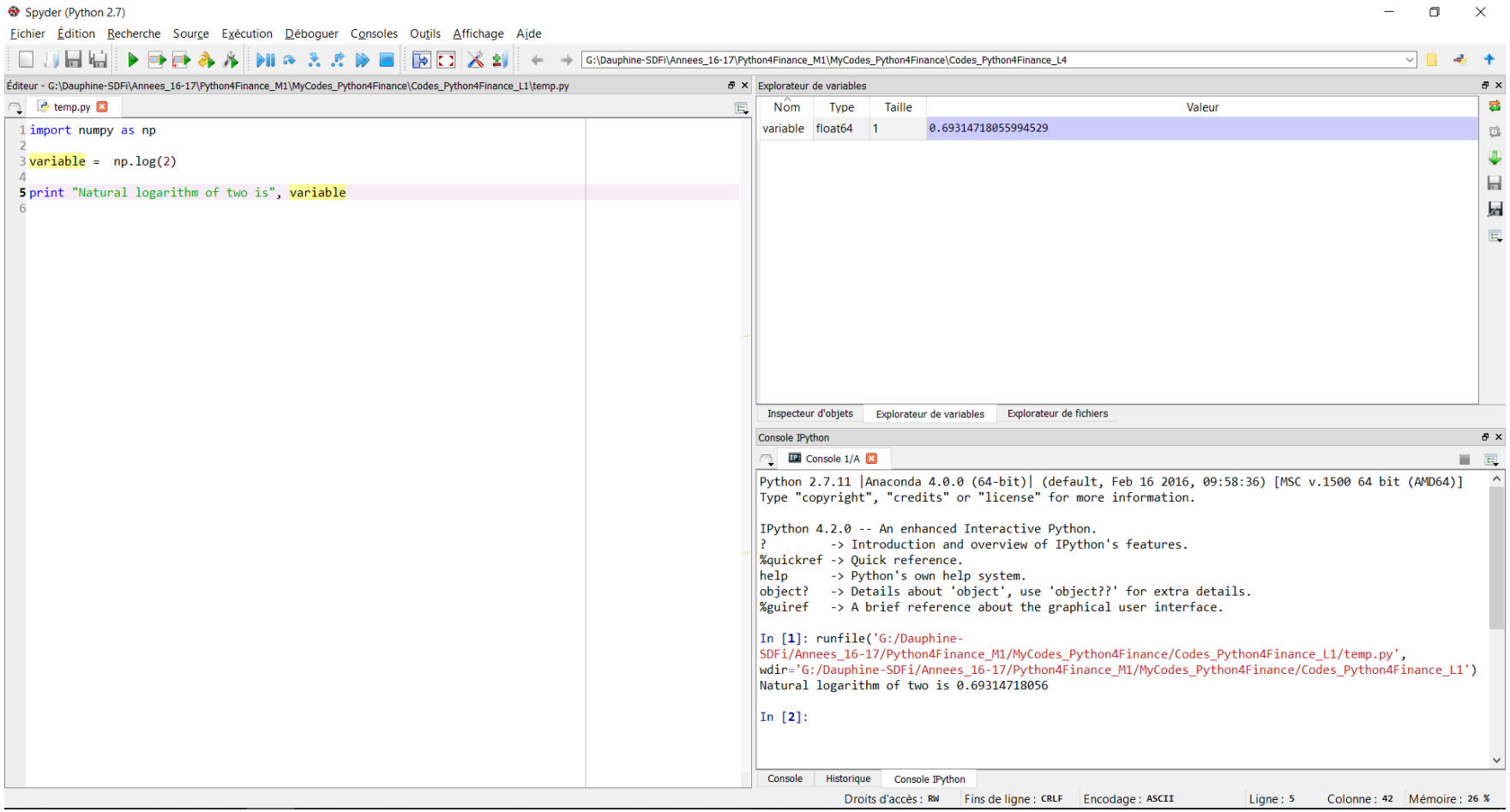
IPython (cross-platform)



Jupyter Notebook (cross-platform, browser based)



Spyder (cross-platform)



Outline

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" $\rightarrow$ "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]: 90.90909090909090909090
```

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`

Excercises

- 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

- Excercise 1

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

- Excercise 2

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

- Excercise 3

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

Outline

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

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

Antigravity

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