

An introduction to Python

Thibaut Lunet, Aitor Pérez

March 19, 2018

How to get Python (+ useful packages ...)

We are going to use the **miniconda** installer, which is cross-platform and provides package management, together with the **spyder** IDE.

1. Go to <https://conda.io/miniconda.html>
(or Google search : "miniconda download")
2. Depending on the operating system, download installer (Python 2.7)
3. Install Python and required packages
 - Mac OS X or Unix:
 - 3.1 Open a terminal
 - 3.2 Run "bash Miniconda[...].sh", and yes for all ...
 - 3.3 Open a new terminal, or run "source ~/.bashrc"
 - 3.4 Run "**conda install spyder numpy scipy matplotlib sympy**"
 - Windows:
 - 3.1 Double-click on the .exe file, and yes for all ...
 - 3.2 Open "conda prompt" terminal (installed with miniconda)
 - 3.3 Run "**conda install spyder numpy scipy matplotlib sympy**"

XXX slkdjlksj

Python vs. Others (Matlab, Fortran, C/C++, ...)

- License-free and open-source (\neq Matlab)
- Huge users community, many (free) packages for many applications
- Extremely easy of use for non-I-love-programming people (\neq Fortran, C/C++)
- Easy interface with other (more-efficient) programming languages
 \Rightarrow computation can be accelerated using Fortran or C/C++ library ...
- Can scale to very large problems (parallel computing, ...)
- Structured and friendly ways for developing library (\neq Matlab)

Python = many advantages, with very few drawbacks !

Functioning principles

XXX dsdskldj

Practical tools

Using python console

```
x - □ lunet@matlnx13: ~/Recherche/Enseignement/python-math/examples
lunet@matlnx13:examples$ python
Python 3.6.4 |Anaconda, Inc.| (default, Jan 16 2018, 18:10:19)
[GCC 7.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> █
```

Running a script

```
x - □ lunet@matlnx13: ~/Recherche/Enseignement/python-math/examples
lunet@matlnx13:examples$ python helloWorld.py
1 + 1 = 2
YAIILLLLLE !
lunet@matlnx13:examples$ █
```

All-in-One solution \Rightarrow **Spyder** !

Hello World!

All the examples and python files are available at:
<https://gitlab.unige.ch/Thibaut.Lunet/python-math>

A first easy step ...

1. Launch Spyder
 - Windows : double click on an icon somewhere ...
 - Mac OS X or Unix : run "spyder" in terminal
2. Discover a wonderful environment #woaaah
3. Go to lower right corner → IPython console
 - write "1+1"
 - press enter ...
4. Go to text editor (middle)
 - write "print('hello world')"
 - save and run the file ...

Basic variables types and operations

Slide codes at: python-math/examples/codeexamples.py

```
# Integer
n = 1
m = 7 % 3 # m = 1

# Float: By default, double precision!
x = 0.5
y = x/7 # y = 0.07142857142857142

# Complex
z = 1+1j
w = z + x + n # Automatic conversion, w = 2.5 + 1j

# String
s = 'salut'
t = 'toi'
r = s + t # r = 'saluttoi'

# Boolean
p = True
q = (n != 1)*p + (n == 1)*(x < 10)*(y >= 0) # q = True = 1
```


Lists

```
# Lists
l = [1, 2, 5, 6]
# Access elements : l[0] = 1, l[2] = 5, l[-1] = 6
# Slice : l[1:3] = [2, 5]

# Nested list
nl = [['vive', 'la'], ['saucisse', 2], 'Toulouse']
# Access sublist element : nl[0] = ['vive', 'la']
# Access final element : nl[0][1] = 'la', nl[1][0] = 'saucisse'

# List comprehension
l1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
l2 = [3 * n + 1 for n in l1 if n % 2 == 0]
# l2 = [7, 13, 19, 25, 31]
```

Dictionary

```
# Dictionaries
d = {'first': 1,
     'second': 'two',
     'third': {'A': [3, 4, 5], 'B': True}}
# Access elements : d['first'] = 1, d['second'] = 'two'
#                  d['third'] = {'A': [3, 4, 5], 'B': True}
#                  d['third']['A'] = [3, 4, 5]
#                  d['third']['A'][1] = 4
#                  d['third']['B'] = True
```

Conditional structures

Tabs matter !

```
# If clause
if 1 == 2:
    print('Tocard')
elif 1 == 0: # Not mandatory
    print('Toujours pas')
else: # Not mandatory
    print("OK d'accord")

# For loop
for i in range(5):
    print('i = {}'.format(i))

# While loop
i = 0
while i < 10:
    print('TAIH000-' + str(i))
    i += 1
    if i == 5:
        break # Allows to escape from the while loop
```

Function definition

```
def add(a, b = 1):  
    return a + b  
  
# add(0.5, 2) = 2.5  
# add(1) = 2  
# add() -> ERROR  
  
# No types  
# Parameters by default  
# Possibility of having a variable number of parameters
```

File I/O

```
infile = open("infile.dat", "r")
# Opens the file for reading

for line in infile:
    print(line)
# We can read each line as a string
# If data is numerical, we can convert via float(line)

infile.close()
# Close file to release memory

outfile = open("outfile.dat", "w")
# Opens the file for writing

s = 'Very important secret message'
outfile.write(s + '\n')
# We can write strings, writing '\n' creates a new line
# If data is numerical, we can convert it via str(x)

outfile.close()
# Close file to release memory
```

Numpy I

```
import numpy as np
# Import the package

v = np.array([1, 2, 3])
# v is the 1-dimensional vector (1, 2, 3)
# We can access and modify its elements via v[0], v[1], v[2]

v.shape # returns (3,)
```



```
A = np.array([[1, 2, 3], [4, 5, 6]])
# A is the 2x3 matrix:
# /1 2 3\
# \4 5 6/
# We can access and modify its elements via A[i, j]

A.shape # returns (2, 3)
```



```
B = np.reshape(A, (3,2))
# B is the 3x2 matrix:
# /1 2\
# |3 4|
# \5 6/
```

Numpy II

We can slice matrices:

```
M = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])
```

```
M[1:2, 0:2]      # Matrix [5, 6]
M[0:2]           # First two rows
M[:, 1:3]        # Second and third columns
```

We already have some built-in matrices:

```
np.zeros((3, 3))    # A 3x3 matrix filled with zeros
np.ones((2, 1))     # A 2x1 matrix filled with ones
np.full((2, 2), 27)  # A 2x2 matrix filled with 27
np.eye(2)           # The identity matrix of dim 2
```

Useful operations:

```
C = B.T            # Transpose
A + C              # Elementwise sum
A - C              # Elementwise difference
A * C              # Elementwise product
A / C              # Elementwise division
np.dot(A, B)       # Matrix product
# And lots of other built-in functions!
```

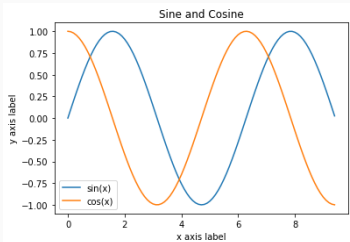
- SciPy Tutorial
 - Introduction
 - Basic functions
 - Special functions (**scipy.special**)
 - Integration (**scipy.integrate**)
 - Optimization (**scipy.optimize**)
 - Interpolation (**scipy.interpolate**)
 - Fourier Transforms (**scipy.fftpack**)
 - Signal Processing (**scipy.signal**)
 - Linear Algebra (**scipy.linalg**)
 - Sparse Eigenvalue Problems with ARPACK
 - Compressed Sparse Graph Routines (**scipy.sparse.csgraph**)
 - Spatial data structures and algorithms (**scipy.spatial**)
 - Statistics (**scipy.stats**)
 - Multidimensional image processing (**scipy.ndimage**)
 - File IO (**scipy.io**)

Matplotlib I

```
import numpy as np
import matplotlib.pyplot as plt
# Import the packages

x = np.arange(0, 3 * np.pi, 0.1)
sinx = np.sin(x)
cosx = np.cos(x)
# We generate the data we want to visualize

plt.plot(x, sinx) # Draws the points (x, sinx)
plt.plot(x, cosx) # Draws the points (x, cosx)
plt.xlabel('x axis label') # Adds a label to the x-axis
plt.ylabel('y axis label') # Adds a label to the y-axis
plt.title('Sine and Cosine') # Adds a title
plt.legend(['sin(x)', 'cos(x)']) # Adds a legend
plt.show() # Actually shows the plot
```



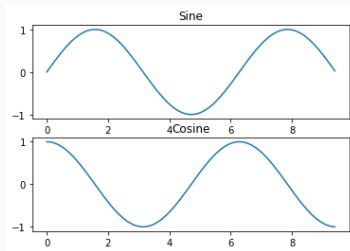
Matplotlib II

We can also display several plots at once:

```
plt.subplot(2, 1, 1)          # 2x1 grid, select the first as active  
plt.plot(x, sinx)  
plt.title('Sine')
```

```
plt.subplot(2, 1, 2)          # 2x1 grid, select the second as active  
plt.plot(x, cosx)  
plt.title('Cosine')
```

```
plt.show()
```



XXX TODO