Introduction to Python

Luis Pedro Coelho
luis@luispedro.org
@luispedrocoelho

European Molecular Biology Laboratory

Lisbon Machine Learning School 2014
Today’s Lecture in Context

- **Today**: basic introduction to Python & Numpy
- During LxMLS, you will implement algorithms “by hand”
- **Tomorrow**: scikit-learn by Andreas Mueller
Python was started in the late 80’s.
It was intended to be both easy to teach and industrial strength.
It is (has always been) open-source.
It has become one of the most widely used languages (top 10).
There are two major versions, currently: 2.7 and 3.4.

We are going to be using 2.7 (but 2.6 should be OK too).
Python Example

print "Hello World"
Task

Average

Compute the average of the following numbers:

1. 10
2. 7
3. 22
4. 14
5. 17
Python example

```python
numbers = [10, 7, 22, 14, 17]

total = 0.0
n = 0.0
for val in numbers:
    total = total + val
    n = n + 1
print total / n
```

Luis Pedro Coelho (@luispedrocoelho) ⋆ Introduction to Python ⋆ #LxMLS (9 / 78)
“Python is executable pseudo-code.”
—Python lore (often attributed to Bruce Eckel)
numbers = [10, 7, 22, 14, 17]

total = 0.0
n = 0.0
for val in numbers:
    total = total + val
    n = n + 1
print total / n
Python Types

Basic Types

- Numbers (integers and floating point)
- Strings
- Lists and tuples
- Dictionaries
A = 1
B = 2
C = 3
print A + B*C

Outputs 7.
A = 1.2
B = 2.4
C = 3.6
print A + B*C

Outputs 9.84.
A = 2
B = 2.5
C = 4.4

print A + B*C

Outputs 22.0.
total = total + n

Can be abbreviated as

total += n
first = 'John'
last = "Doe"
full = first + " " + last

print full
Python Types: Strings

```python
first = 'John'
last = "Doe"
full = first + " " + last

print full
```

Outputs *John Doe*. 
What is a String Literal

- Short string literals are delimited by (”) or (’).
- Short string literals are one line only.
- Special characters are input using escape sequences. (\n for newline,...)

```
multiple = 'He: May I?\nShe: No, you may not.'
alternative = "He: May I?\nShe: No, you may not."
```
We can input a long string using triple quotes (""" or """"""") as delimiters.

```python
long = """"Tell me, is love
Still a popular suggestion
Or merely an obsolete art?

Forgive me, for asking,
This simple question,
I am unfamiliar with his heart."""
```
courses = [ 'PfS', 'Political Philosophy' ]

print "The the first course is", courses[0]
print "The second course is", courses[1]

Notice that list indices start at 0!
Python Types: Lists

```python
mixed = ['Banana', 100, ['Another', 'List'], []]
print(len(mixed))
```
Python Types: Lists

```
fruits = ['Banana', 'Apple', 'Orange']
fruits.sort()
print fruits

Prints ['Apple', 'Banana', 'Orange']
```
emails = {'Luis': 'lpc@cmu.edu',
          'Mark': 'mark@cmu.edu'}

print "Luis's email is", emails['Luis']

emails['Rita'] = 'rita@cmu.edu'
student = 'Rita'
average = gradeavg(student)
if average > 0.7:
    print student, 'passed!'
    print 'Congratulations!!'
else:
    print student, 'failed. Sorry.'
Unlike almost all other modern programming languages, Python uses **indentation** to delimit blocks!

```python
if <condition>:
    statement 1
    statement 2
    statement 3
next statement
```
## Convention

1. Use 4 spaces to indent.
2. Other things will work, but confuse people.
Conditionals

Examples

- `x == y`
- `x != y`
- `x < y`
- `x < y < z`
- `x in lst`
- `x not in lst`
if <condition 1>:
    do something
if condition 2>:
    nested block
else:
    nested else block
elif <condition 1b>:
    do something
For loop

```python
students = ['Luis', 'Rita', 'Sabah', 'Mark']
for st in students:
    print(st)
```
While Loop

\[
\text{while } \langle \text{condition} \rangle : \\
\text{statement1} \\
\text{statement2}
\]
for i in range(5):
    print i

prints

0
1
2
3
4

This is because range(5) is the list [0,1,2,3,4].
rita_enrolled = False
for st in students:
    if st == 'Rita':
        rita_enrolled = True
        break
Booleans

• Just two values: True and False.
• Comparisons return booleans (e.g., \( x < 2 \))

Conditions

• When evaluating a condition, the condition is converted to a boolean:
• Many things are converted to False:
  1. \([\] \) (the empty list)
  2. \(\{} \) (the empty dictionary)
  3. """" (the empty string)
  4. 0 or 0.0 (the value zero)
  5. ...
• Everything else is True or not convertible to boolean.
A = []
B = [1, 2]
C = 2
D = 0

if A:
    print 'A is true'
if B:
    print 'B is true'
if C:
    print 'C is true'
if D:
    print 'D is true'
Numbers

Two Types of Numbers

1. Integers
2. Floating-point

Operations

1. Unary Minus: \(-x\)
2. Addition: \(x + y\)
3. Subtraction: \(x - y\)
4. Multiplication: \(x \times y\)
5. Exponentiation: \(x^{**} y\)
Division

What is 9 divided by 3?
What is 10 divided by 3?
Division

What is 9 divided by 3?
What is 10 divided by 3?

Two types of division

1. Integer division: $x \div y$
2. Floating-point division: $x / \text{float}(y)$
```python
def double(x):
    '''
    y = double(x)
    Returns the double of x
    '''
    return 2*x
```

Functions
Functions

A=4

print double(A)
print double(2.3)
print double(double(A))
def greet(name, greeting='Hello '):
    print greeting, name

greet('Mario')
greet('Mario', 'Goodbye')
Defining a class

**Boat Class**

We define a Boat class, with two values, latitude & longitude, and five methods:

1. `move_north`, `move_south`, `move_east`, `move_west`
2. `distance`
Defining & Calling Methods

Defining a method

class Boat(object):
    def __init__(self, lat=0, long=0):
        self.latitude = lat
        self.longitude = long

    def move_north(self, dlat):
        self.latitude += dlat

Calling a Method

obj = Boat()

obj.method(arg1, arg2)
class Boat(object):
    def __init__(self, lat=0, long=0):
        self.latitude = lat
        self.longitude = long

    def move_north(self, dlat):
        self.latitude += dlat

- __init__: special name (constructor)
- self: the object itself (this in many other languages)
- Instance variables are defined at first use
class ScientificBoat(object):
    def __init__ (self, lat=0, long=0):
        self.latitude = lat
        self.longitude = long

    def move_north(self, dlat):
        ...

Luis Pedro Coelho (@luispedrocoelho)  ⋆  Introduction to Python  ⋆  #LxMLS (43 / 78)
Numpy
numpy.array or numpy.ndarray.

Multi-dimensional array of numbers.
import numpy as np
A = np.array(
    [
        [0, 1, 2],
        [2, 3, 4],
        [4, 5, 6],
        [6, 7, 8]]
)
print A[0, 0]
print A[0, 1]
print A[1, 0]
import numpy as np
A = np.array([
    [0, 1, 2],
    [2, 3, 4],
    [4, 5, 6],
    [6, 7, 8]]
)
print A[0, 0]
print A[0, 1]
print A[1, 0]

0
1
2
Why do we need numpy?

```python
import numpy as np
lst = [0., 1., 2., 3.]
arr = np.array([0., 1., 2., 3.])
```
A Python List of Numbers

float 0.0
float 1.0
float 2.0
float 3.0
A Numpy Array of Numbers

<table>
<thead>
<tr>
<th>float</th>
<th>0.0</th>
<th>1.0</th>
<th>2.0</th>
<th>3.0</th>
</tr>
</thead>
</table>

Advantages

- Less memory consumption
- Faster
- Work with (or write) code in other languages (C, C++, Fortran...)
Matrix-vector multiplication

```python
A = np.array(
    [
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]
    ]
)

v = np.array([1, 5, 2])

print(np.dot(A, v))
```
Matrix-vector multiplication

\[
\begin{align*}
A &= \text{np.array}([ \\
&\quad [1, 0, 0], \\
&\quad [0, 1, 0], \\
&\quad [0, 0, 1]]) \\
V &= \text{np.array}([1, 5, 2]) \\
\text{print } \text{np.dot}(A, V) \\
\end{align*}
\]

\[
[1 5 2]
\]
Matrix-Matrix and Dot Products

\[
\begin{pmatrix}
1 & 1 \\
1 & -1
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix} =
\begin{pmatrix}
1 & 1 \\
-1 & 1
\end{pmatrix}
\]
Matrix-Matrix and Dot Products

\[
\begin{pmatrix}
1 & 2 \\
-1 & 3
\end{pmatrix}
\begin{pmatrix}
3 \\
1
\end{pmatrix}
= 1 \cdot 3 + (-1) \cdot 2 = 1.
\]

This is a vector inner product (aka dot product)

\[
< \vec{x}, \vec{y} > = \vec{x} \cdot \vec{y} = \vec{x}^T \vec{y}.
\]
v0 = np.array([1, 2])
v1 = np.array([3, -1])

r = 0.0
for i in xrange(2):
    r += v0[i]*v1[i]
print r

print np.dot(v0, v1)
A0 = np.array([[1, 2], [2, 3]])
A1 = np.array([[0, 1], [1, 0]])

print np.dot(A0, A1)

\[
\begin{pmatrix}
0 & 2 \\
2 & 3
\end{pmatrix}
\begin{pmatrix}
0 & 1 \\
1 & 0
\end{pmatrix}
\]
Some Array Properties

```python
import numpy as np
A = np.array([[0, 1, 2], [2, 3, 4], [4, 5, 6], [6, 7, 8]])
print(A.shape)
print(A.size)
```
Some Array Functions

... 

```python
print A.max()
print A.min()
```

- `max()`: maximum
- `min()`: minimum
- `ptp()`: spread (max - min)
- `sum()`: sum
- `std()`: standard deviation
- ...

Luis Pedro Coelho (@luispedrocoelho)
Other Functions

- np.exp
- np.sin
- ...

All of these work element-wise!
Arithmetic Operations

```python
import numpy as np
A = np.array([0, 1, 2, 3])
B = np.array([1, 1, 2, 2])

print A + B
print A * B
print A / B
```
import numpy as np
A = np.array([0, 1, 2, 3])
B = np.array([1, 1, 2, 2])

print A + B
print A * B
print A / B

[1 2 4 5]
[0 1 4 6]
[0 1 1 1]
Numpy Dtypes

- All members of an array have the same type
- Either integer or floating point
- Defined when you first create the array

```python
A = np.array([0, 1, 2])
B = np.array([0.5, 1.1, 2.1])

A * 2.5
B * 2.5

print A
print B

[0, 2.5]
[1.25, 2.75, 5.25]
```
A = np.array([0, 1, 2], dtype=np.int16)
B = np.array([0, 1, 2], dtype=np.float32)

- np.int8, np.int16, np.int32
- np.uint8, np.uint16, np.uint32
- np.float32, np.float64
- np.bool
Object Construction

```python
import numpy as np
A = np.array([0, 1, 1], np.float32)
A = np.array([0, 1, 1], float)
A = np.array([0, 1, 1], bool)
```
Reduction

```python
A = np.array([
    [0, 0, 1],
    [1, 2, 3],
    [2, 4, 2],
    [1, 0, 1]])
print A.max(0)
print A.max(1)
print A.max()
```

prints

```
[2, 4, 3]
[1, 3, 4, 1]
4
```

The same is true for many other functions.
import numpy as np
A = np.array(
    [
        [0, 1, 2],
        [2, 3, 4],
        [4, 5, 6],
        [6, 7, 8]
    ]
)
print A[0]
print A[0].shape
print A[1]
print A[:, 2]
Slicing

```python
import numpy as np
A = np.array([
    [0, 1, 2],
    [2, 3, 4],
    [4, 5, 6],
    [6, 7, 8]
])
print A[0]
print A[0].shape
print A[1]
print A[:, 2]

[0, 1, 2]
(3,)
[2, 3, 4]
[2, 4, 6, 8]
```
import numpy as np

A = np.array([
    [0, 1, 2],
    [2, 3, 4],
    [4, 5, 6],
    [6, 7, 8]
])

B = A[0]
B[0] = -1
print A[0, 0]
def double(A):
    A *= 2

A = np.arange(20)
double(A)
Pass is By Reference

def double(A):
    A *= 2

A = np.arange(20)
double(A)

A = np.arange(20)
B = A.copy()
Logical Arrays

```python
A = np.array([-1, 0, 1, 2, -2, 3, 4, -2])
print (A > 0)
```
Logical Arrays II

A = np.array([-1, 0, 1, 2, -2, 3, 4, -2])
print((A > 0) & (A < 3)).mean()

What does this do?
Logical Indexing

\[ A[A < 0] = 0 \]

or

\[ A *= (A > 0) \]
Logical Indexing

print 'Mean of positives', A[A > 0].mean()
Some Helper Functions

Constructing Arrays

A = np.zeros((10, 10), dtype=np.int8)
B = np.ones(10)
C = np.arange(100).reshape((10, 10))
...

Multiple Dimensions

img = np.zeros((1024, 1024, 3), dtype=np.uint8)
http://docs.scipy.org/doc/
Matplotlib & Spyder
Matplotlib is a plotting library.

- Very flexible.
- Very active project.
Example I

```python
import numpy as np
import matplotlib.pyplot as plt
X = np.linspace(-4, 4, 1000)
plt.plot(X, X**2*np.cos(X**2))
plt.savefig('simple.pdf')

y = x^2 \cos (x^2)
```
Resources

- Numpy+scipy docs: http://docs.scipy.org
- Matplotlib: http://matplotlib.sf.net
- Python docs: http://docs.python.org

- These slides are available at http://luispedro.org/talks/2014
- I’m available at luis@luispedro.org
  @luispedrocoelho on twitter
Thank you.