Introduction to Python

Luis Pedro Coelho

Institute for Molecular Medicine (Lisbon)

Lisbon Machine Learning School II

Python Language History

- 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).

Python Versions

Python Versions

- There are two major versions, currently: 2.7 and 3.2.
- 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

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

sum = 0.0
n = 0.0
for val in numbers:
    sum = sum + val
    n = n + 1
return sum / n
```

"Python is executable pseudo-code."
—Python lore (often attributed to Bruce Eckel)

Programming Basics

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

sum = 0.0
n = 0.0
for val in numbers:
    sum = sum + val
    n = n + 1
return sum / n
```

Python Types

Basic Types

- Numbers (integers and floating point)
- Strings
- Lists and tuples
- Dictionaries

Python Types: Numbers I: Integers

```
A = 1
B = 2
C = 3
print A + B*C
```

Outputs 7.

Python Types: Numbers II: Floats

```
A = 1.2
B = 2.4
C = 3.6
print A + B*C
```

Outputs 9.84.

Python Types: Numbers III: Integers & Floats

```
A = 2
B = 2.5
C = 4.4
print A + B*C
```

Outputs 22.0.

Composite Assignment

```
total = total + n
```

Can be abbreviated as

```
total += n
```

Python Types: Strings

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

print full
```

Python Types: Strings

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

print full
```

Outputs John Doe.

Python Types: String Rules

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."
```

Python Types: Long Strings

We can input a long string using triple quotes (”’ or ”””) as delimiters.

```
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.'''
```

Python Types: Lists

```
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']

Python Types: Dictionaries

```
emails = { 'Luis' : 'lpc@cmu.edu',
           'Mark' : 'mark@cmu.edu' }
print "Luis's email is", emails['Luis']

emails['Rita'] = 'rita@cmu.edu'
```

Python Control Structures

```
student = 'Rita'
average = gradeavg(student)
if average > 0.7:
    print student, 'passed!'
    print 'Congratulations!!'
else:
    print student, 'failed. Sorry.'
```

Python Blocks

Unlike almost all other modern programming languages, Python uses indentation to delimit blocks!

```
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

Nested Blocks

```
if <condition 1>:
    do something
    if condition 2>:
        nested block
    else:
        nested else block
elif <condition 1b>:
    do something
```

For loop

```
students = ['Luis','Rita','Sabah','Mark']
for st in students:
    print st
```

While Loop

```
while <condition>:
    statement1
    statement2
```

Other Loopy Stuff

```
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].

Break

```
rita_enrolled = False
for st in students:
    if st == 'Rita':
        rita_enrolled = True
        break
```

Conditions & Booleans

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.

Conditions Example

```
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 * y
5. Exponentiation: x ** y

Division

Division

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

Division

Division

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

Two types of division

1. Integer division: x // y

Functions

```python
def double(x):
    '''
    y = double(x)

    Returns the double of x
    '''
    return 2*x
```

Functions

```
A=4
print double(A)
print double(2.3)
print double(double(A))
```

- Vectors

$$[0, 1.2, -1.2, 4] \in \mathrm{R}^4$$

- Matrices (operators)

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

- Multiplication of vectors & Matrices

Vector Space

- Addition operation

$$[1, 2] + [2, 3] = [3, 5]$$

- Multiplication by a scalar

$$4 \cdot [2, 0, 1] = [8, 0, 4]$$

Matrix as an Operator

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix} \begin{pmatrix} x_0 \\ x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_0 A_{00} + x_1 A_{01} + x_2 A_{02} \\ x_0 A_{10} + x_1 A_{11} + x_2 A_{12} \\ x_0 A_{20} + x_1 A_{21} + x_2 A_{22} \end{pmatrix}$$

Matrix Transpose

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}^{T} = \begin{pmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{pmatrix}$$

Matrix Transpose

$$\begin{pmatrix} A_{00} & A_{01} & A_{02} \\ A_{10} & A_{11} & A_{12} \\ A_{20} & A_{21} & A_{22} \end{pmatrix}^{T} = \begin{pmatrix} A_{00} & A_{10} & A_{20} \\ A_{01} & A_{11} & A_{21} \\ A_{02} & A_{12} & A_{22} \end{pmatrix}$$

$$\begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}^{T} = \begin{pmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{pmatrix}$$

Numeric Python: Numpy

Numpy

Basic Type

numpy.array or numpy.ndarray.

Multi-dimensional array of numbers.

numpy example

```
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]
```

numpy example

```
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]
```

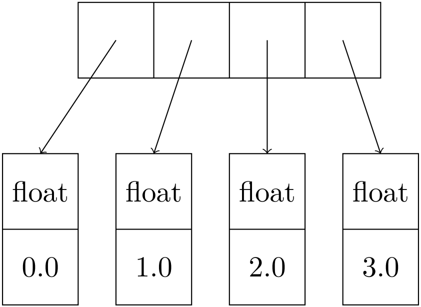
```
0
1
2
```

Why Numpy?

Why do we need numpy?

```
import numpy as np
lst = [0.,1.,2.,3.]
arr = np.array([0.,1.,2.,3.])
```

A Python List of Numbers



A Numpy Array of Numbers

float	0.0	1.0	2.0	3.0

Numpy Arrays

Advantages

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

Matrix-vector multiplication

```
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

```
A = np.array([
        [1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]])
v = np.array([1, 5, 2])

print np.dot(A,v)
```

[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

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

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)
```

```python
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

```
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

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

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

Other Functions

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

All of these work element-wise!

Arithmetic Operations

```
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
```

Arithmetic Operations

```
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

```
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]
```

```python
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

```
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

```
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.

Slicing

```
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

```
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]
```

Slices Share Memory!

```
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]
```

Pass is By Reference

```
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

```
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),dype=np.uint8)
```

Documentation

http://docs.scipy.org/doc/

Last Section

Matplotlib & Spyder

Matplotlib

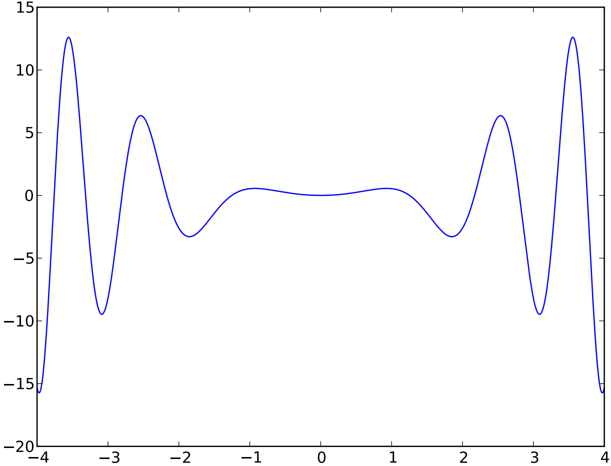
- Matplotlib is a plotting library.
- Very flexible.
- Very active project.

Example I

```
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')
```

$$y = x^2 \cos\left(x^2\right)$$

Example I



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/2012
- I'm available at luis@luispedro.org

Thank you.