

# Introduction to Python

Luís Pedro Coelho

Programming for Scientists

February 5, 2009



University of Pittsburgh

**Carnegie Mellon**

# Installing Python

Let's digress for a moment discussing the 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 and has become one of the most widely used languages (top 10).

Unlike *Matlab*, Python does not come as a single programme.  
This is a cultural difference.

# Example of Python IDEs

## Integrated Development Environments

- Eclipse
- Stani (Stan's Python editor)
- DrPython
- Komodo Edit
- Unix world: ipython + (vim or emacs or nano) + command-line
- ...

Links for the above are on the webpage under “notes for lab session 1”.

# Python Versions

## Python Versions

- The current version of Python is **2.6** (October 2008).
- This class assumes you have 2.5 or 2.6.
- There are some small differences, but only one difference is important (we'll get to that).

## Python 3.0

We are going to ignore Python 3!

# Python Example

```
print "Hello World"
```

## Running Python

- 1 From a file
- 2 Interactively

# Back to the Python Language

Let's look at the language itself.

## 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
```

```
n = 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
```

```
n = 0
```

```
for val in numbers:
```

```
    sum = sum + val
```

```
    n = n + 1
```

```
return sum / n
```

## 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.**

## 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!

## Indices Start at Zero

- First element is denoted *list[0]*, last element is *list[N-1]*.
- Think of them as **offsets**.

# Python Types: Lists

```
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.'
```

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

```
if <condition>:  
    statement 1  
    statement 2  
    statement 3  
next statement
```

# Adhere to Convention

## Principle

- 1 Learn about the conventions of the technology you're using.
- 2 Adhere to them.

Use 4 spaces to indent.

## 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

```
emails = { 'Luis' : 'lpc@cmu.edu',  
           'Mark' : 'mark@cmu.edu',  
           'Rita' : 'rsmith@pitt.edu' }  
students = ['Luis', 'Rita', 'Sabah', 'Mark']  
for st in students:  
    if st not in emails.keys():  
        print 'We need to ask', st, 'for her email'
```

# While Loop

```
i = 0
while i < len(students):
    if is_enrolled(students[i]):
        i += 1
    else:
        del students[i]
```

# Other Loopy Stuff

```
for i in range(5):  
    print i
```

prints

0  
1  
2  
3  
4

# Range

```
R = range(5)
```

```
R2 = [0, 1, 2, 3, 4]
```

R and R2 are identical lists.

# Xrange: Faster looping

```
for i in xrange(5):  
    print i
```

prints

0

1

2

3

4

# Break

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

# Continue

```
for st in students:  
    if turned_in_homework(st):  
        continue  
    print '%s did not turn in his homework on time' % st  
    <statement>  
    <statement>  
    <statement>
```

Notice the string formatting on the last line!

# String Formatting

```
"string with %s placeholders %s" % (arg1, arg2)
```

- The %s placeholders are replaced by the passed arguments with the % operator.
- Use %% to get a %.

# 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'
```

## 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?

## Two types of division

- 1 Floating-point division:  $x / y$  (in Python 2.6)
- 2 Integer division:  $x // y$

## Python 2.5 vs. 2.6

- This is the one important difference between the two versions of Python:
  - 1 In Python 2.5,  $x / y$  means **integer division** by default.
  - 2 In Python 2.6, it means **floating-point division**.
- We can get the 2.6 behaviour by adding *from \_\_future\_\_ import division* at the top of your file

# Importing

```
A = 10  
B = 3  
print A / B
```

```
from __future__ import division
```

```
A = 10  
B = 3  
print A / B
```

`from __future__` must be at the start of a file!

# Functions

```
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))
```