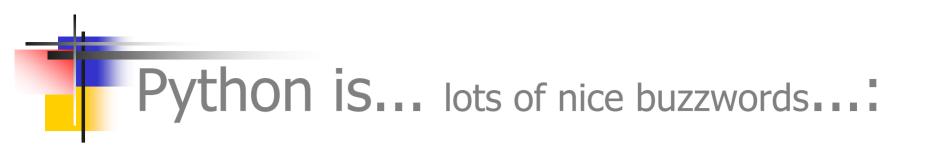
1

Python (& Jython) introduction

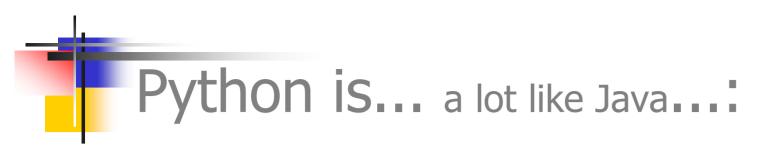
for C++ and Java programmers

Alex Martelli <alex@strakt.com>

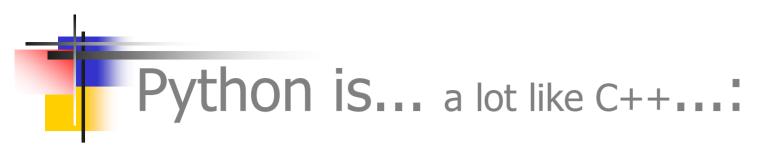
STRAKT



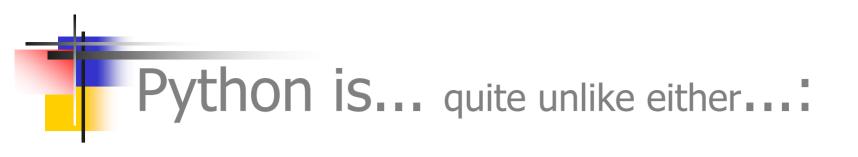
- a Very-High-Level Language (VHLL)
- clean, spare syntax
- simple, regular, orthogonal semantics
- high-productivity
- open-source, cross-platform
- object-oriented
- **...**



- compiler to bytecode + VM/interpreter
 - however, compilation is implicit ("auto-make")
- everything (in 2.2) inherits from object
- consistent "object-reference" semantics
 - assignment, argument-passing, ...
 - applies to numbers too (immutable, like strings)
- large, powerful standard library
- introspection, serialization, threading...



- multi-paradigm
 - object-oriented, procedural, ...
- multiple inheritance
- operator overloading
- signature-based polymorphism
 - as if "everything was a template"... w/ clean syntax
- choices, choices everywhere
 - GUIs, Web server frameworks, COM/Corba/...



- strong but dynamic typing
 - objects have (strong) types, names don't
 - no declarations -- only statements
- clean syntax, minimal "chart-junk"
 - blocks have no { } -- just indentation
 - if/while have no ()
- most everything is a first-class object
 - including classes, functions, modules, packages...

Python versions/releases

- Classic Python: currently 2.2 -> 2.3
 - implemented in 1990-level ISO C
- Jython: currently 2.1 -> (2.2/2.3)
 - implemented as 100% pure Java
 - deploy just like Java, on a JVM
 - transparently use/extend/implement arbitrary
 Java classes and interfaces / compile to Java / ...
- Others: experimental/research level
 - Python.NET , PyPy , Vyper (O'CAML), ...

Python resources on the net

- http://www.python.org
 - just about everything: downloads, docs, mailing lists, SIGs, pointers to [whatever], ...
 - http://www.python.org/ftp/python/2.3/Python-2.3a2.exe
- http://www.jython.org
 - mostly Jython-specific stuff
- news:comp.lang.python
 - any kind of question, request, discussion
- http://www.google.com (no, really!!!)



- interactive interpreter (text and IDLE)
 - mostly for trying things out, or as a calculator
 - prompts with >>>, shows expressions' results
- program files (afile.py, afile.pyc, ...)
 - for most uses; compilation is automatic
- assignment (simplest form):
 - name = <any expression>
 - creates name if needed, binds it to the value
 - names are not declared, and have no type per se

assignments, print

```
myvar = 'hello'  # creates a name
myvar = 23
                  # rebinds name
question = answer = 42
myvar, answer = answer, myvar
print myvar, answer, question
42, 23, 42
if myvar<20: myvar = 10 # not executed
if myvar>40: myvar = 50 # executed
print myvar
50
```

conditional statements

```
# 'if' guards a suite
if question>30:
   question = 20
                         the suite is shown
   x = 33
                         by its indentation
                      optional 'else'
else:
                         indentation again
   x = 99
if x<30: myvar = 10
                          # not met
elif x<40: myvar = 20
                          # met
elif x<50: myvar = 40
                          # not evaluated
else: myvar = 40
                          # this neither
print x, question, myvar
33 20 20
```

comparisons, tests, truth

```
equality, identity: == != is is not order: < > <= >= containment: in not in comparisons chain: 5<x<9 a==b==c
```

```
false: any ==0, "", None, empty containers
true: every other value
in Python 2.2.1 and higher:
   False==0, True==1
   bool(x) gives True or False
```

short-

short-circuit and/or; not

```
and/or short-circuit and return either operand:
x = y and z is like: if y: x=z
                     else: x=y
x = y or z is like: if y: x=y
                     else: x=z
print 0 and 0j, 0j and 0, 0 or 0j, 0j or 0
0 0j 0j 0
x = not y is like: if y: x=0 \# True (2.2)
                     else: x=1 \# False
print not 0, not 1, not 0, not 1, \# 2.3
True False True False
```



```
int (usually 32-bit) and long (unlimited precision):
  print 2**100
 1267650600228229401496703205376
float (usually 64-bit IEEE):
  print 2**100.0
 1.26765060023e+030
complex (float real and imag parts):
  print 2**100.0j
 (0.980130165912+0.19835538276j)
```



```
add, subtract, multiply, power: + - * **
division (true, truncating, mod): / // %
bitwise and shift: ~ & | ^ << >>
built-in functions: abs divmod max min pow round
conversions: complex float int long
```

print 2**100%999, pow(2,100,999) *160 160*



```
while myvar>10: myvar -= 7
print myvar
3
for i in 0, 1, 2, 3: print i**2,
0 1 4 9
for i in range(4): print i**2, # "UBX"
0 1 4 9
```

while and for normally control suites (blocks)
 may contain break, continue
optional else clause == "natural termination"

files (example: copying)

```
fin = open('in','r') # or just open('in')
fou = open('ou','w') # 'a', 'r+', 'wb'...
fou.write(fin.read())
                                 # or:
data=fin.read(); fou.write(data) # or:
fou.writelines(fin.readlines()) # or:
for line in fin: fou.write(line) # 2.2/+
fin.close()
              # good practice, but only
              # "mandatory" in Jython
fou.close()
```

strings (example: file-listing)

```
# in 2.3:
for lineNumber, lineText in enumerate(fin):
    fou.write('Line number %s: %s'
        % (lineNumber+1, lineText))
# or, in 2.2:
lineNumber = 0
for lineText in fin:
    lineNumber += 1
# or, in 2.1:
lineNumber = 0
for lineText in fin.readlines():
```

16

strings are sequences

```
for c in 'ciao': print c,
ciao
print len('cip'),'i' in 'cip','x' in 'cip'
3 True False # or 3 1 0 in 2.2
# also: 'ia' in 'ciao' -- but, 2.3 only
print 'Oxford'[2], 'Oxford'[1:4]
f xfo
print 'ci'+'ao', 'cip'*3, 4 * 'pic'
ciao cipcipcip picpicpicpic
```

lists are heterogeneous vectors

```
x = [1, 2, 'beboop', 94]
x[1] = 'plik'
             # lists are mutable
print x
[1, 'plik', 'beboop', 94]
x[1:2] = [6,3,9] # can assign to slices
print x
[1, 6, 3, 9, 'beboop', 94]
print [it*it for it in x[:4]]
[1, 36, 9, 81]
```

lists are also sequences

```
print x
[1, 6, 3, 9, 'beboop', 94]
for it in x: print it,
1 6 3 9 beboop 94
print len(x), 6 in x, 99 in x
6 True False # or 3 1 0 in 2.2
print x[2], x[1:5]
3 [6, 3, 9, 'beboop']
print [1]+[2], [3,4]*3
[1, 2] [3, 4, 3, 4, 3, 4]
```

seq

Sequence indexing and slicing

```
x = 'helloworld'
print x[1], x[-3]
er
print x[:2], x[2:], x[:-3], x[-3:]
he lloworld hellowo rld
print x[2:6], x[2:-3], x[5:99]
11ow 11owo world
# step is only allowed in Python 2.3:
print x[::2], x[-3:4:-1]
hlool row
```

sequ

sequence packing/unpacking

```
x = 2, 3, 9, 6
                    # tuple (immutable)
print x
(2, 3, 9, 6)
a, b, c, d = x # unpacking
print c, b, d, a
9 3 6 2
RED, YELLOW, GREEN = range(3) # enum-like
a, b, c, d = 'ciao' # unpacking
print c, b, d, a
a i o c
```

string methods

```
x = 'ciao'
print x.upper(), x.title(), x.isupper()
CIAO Ciao False
print x.find('a'), x.count('a'), x.find('z')
21 - 1
print x.replace('a','e')
cieo
print x.join('what')
wciaohciaoaciaot
```

list methods

```
x = list('ciao')
print x ['c', 'i', 'a', 'o']
print x.sort()
None
print x
['a', 'c', 'i', 'o']
print ''.join(x)
acio
x.append(23); print x
['a', 'c', 'i', 'o', 23]
```

list comprehensions

```
[ <expr> for v in seq ]
[ <expr> for v in seq if <cond> ]
```

squares of primes between 3 and 40 def p(x):

return [n for n in range(2,x) if x%n==0] print [x*x for x in range(3,40) if not p(x)] [9, 25, 49, 121, 169, 289, 361, 529, 841, 961, 1369]

refere

reference semantics

```
x = ['a', 'b', 'c']
y = x
x[1] = 'zz'
print x, y
['a', 'zz', 'c'] ['a', 'zz', 'c']
# explicitly ask for a copy if needed:
y = list(x)
                    # or x[:], or...
x[2] = 9999
print x, y
['a', 'zz', 9999] ['a', 'zz', 'c']
```

dicts are mappings

```
x = \{1:2, 'beboop':94\}
x[1] = 'plik' # dicts are mutable
print x
{1:'plik', 'beboop':94}
x['z'] = [6,3,9] # can add new items
print x
{1:'p7ik', 'z':[6, 3, 9], 'beboop':94}
print dict([ (i,i*i) for i in range(4) ])
{0:0, 1:1, 2:4, 3:9}
```

dicts keys

```
Must be hashable (normally: immutable)...:
X = \{\}
x[[1,2]] = 'a list'
TypeError: list objects are unhashable
x[\{1:2\}] = 'a dict'
TypeError: dict objects are unhashable
x[1,2] = 'a tuple' # ok, tuple's hashable
x[0j] = 'a complex' # all numbers are OK
x[0.0] = 'a float' # **however**...:
                    # 0==0.0==0j, so...:
print x[0]
a float
```

dicts are not sequences, but...:

```
print x
{1:'p7ik', 'z':[6, 3, 9], 'beboop':94}
for it in x: print it, # in 2.2 / 2.3
1 z beboop
for it in x.keys(): print it,
1 z beboop
print len(x), 'z' in x, 99 in x
3 True False # or 3 1 0 in 2.2
```

dict methods

```
print x.get(1), x.get(23), x.get(45,'bu')
plik None bu
print x
{1: 'p7ik', 'z':[6,3,9], 'beboop':94}
print x.setdefault(1,'bah')
p7ik
print x.setdefault(9,'W')
W
print x
{1: 'p7ik',9: 'W', 'z': [6,3,9], 'beboop':94}
```

examp

example: indexing a textfile (2.3)

```
# build a word -> line numbers mapping
idx = \{\}
for n, line in enumerate(open('some.txt')):
    for word in line.split():
        idx.setdefault(word,[]).append(n)
# display by alphabetically-sorted word
words = idx.keys(); words.sort()
for word in words:
    print "%s:" % word,
    for n in idx[word]: print n,
    print
```

example

example: C++ equivalent

```
#include <string>
                                                  for(std::map<std::string, std::vector<int> >
#include <iostream>
                                                            ::iterator i = idx.begin();
#include <sstream>
                                                            i != idx.end(); ++i) {
#include <map>
                                                       std::cout << i->first << ": ":
#include <vector>
                                                       for(std::vector<int>
int main()
                                                                 ::iterator j = i->second.begin();
   std::map<std::string, std::vector<int> > idx;
                                                                j != i->second.end(); ++j) {
   std::string line;
                                                            std::cout << ' ' << *j;
   int n = 0;
   while(getline(std::cin, line)) {
                                                       std::cout << "\n";</pre>
       std::istringstream sline(line);
       std::string word;
                                                   }
       while(sline >> word) {
           idx[word].push_back(n);
                                                   return 0:
       n += 1;
```

on KJB, 4.4MB: C++ 8.5/17.40 (opt. 7.38/15.01) Python 5.4/11.22 (opt. 3.85/8.09)



```
def sumsquares(x, y): return x**2+y**2
print sumsquares(1, 2)
def sq1(x, y=1): return sumsquares(x, y)
print sq1(1, 2), sq1(3)
5 10
def ssq(*args):
               # varargs-like
    total = 0
    for arg in args: total += arg**2
    return total
```

functions support lexical closure

```
def makeAdder(addend):
    def adder(augend):
        return augend+addend
    return adder
ad23 = makeAdder(23)
ad42 = makeAdder(42)
print ad23(100),ad42(100),ad23(ad42(100))
123 142 165
```



```
class act:
                         # class attribute
    cla = []
    def __init__(self): # constructor
        self.ins = {} # inst. attribute
    def meth1(self, x):
        self.cla.append(x)
    def meth2(self, y, z):
        self.ins[y] = z
# calling the class creates an instance:
ex1 = act()
ex2 = act()
```

classes and instances

```
print ex1.cla, ex2.cla, ex1.ins, ex2.ins
ex1.meth1(1); ex1.meth2(2, 3)
ex2.meth1(4); ex2.meth2(5, 6)
print ex1.cla, ex2.cla, ex1.ins, ex2.ins
\[ \int 1. 47 \int 1. 47 \{2: 3\} \{5: 6\} \]
print ex1.cla is ex2.cla
True
```



```
class sub(act):
    def meth2(self, x, y=1): # override
        act.meth2(self, x, y) # supercall
class stutter(list):
                               # 2.2/2.3
    def append(self, x):
        for i in 1,2:
            list.append(self, x)
class dataoverride(sub):
    cla = stutter()
```

new-style classes (2.2, 2.3)

```
class ns(object):
 def hlo(): return 'hello'
  hlo = staticmethod(hlo)
  def hi(cls): return 'hi,%s'%cls.__name___
  hi = classmethod(hi)
class sn(ns): pass
print ns.hlo(), sn.hlo(), ns.hi(), sn.hi()
hello hello hi,ns hi,sn
x = ns(); y = sn()
print x.hlo(), y.hlo(), x.hi(), y.hi()
hello hello hi,ns hi,sn
```

properties (2.2, 2.3)

```
class evener(object):
 def ___init__(self, num): self.x = num
 def getNum(self): return self.x*2
 def setNum(self, num): self.x = num//2
 num = property(getNum, setNum)
x = evener(23); print x.num
22
x.num = 27.12; print x.num
26.0
```

operator overloading

```
class faker:
    def __add__(self, other): return 23
    def __mul__(self, other): return 42
x = faker()
print x+5, x+x, x+99, x*12, x*None, x*x
23 23 23 42 42 42
```

Can overload: all arithmetic, indexing/slicing, attribute access, length, truth, creation, initialization, copy, ... but **NOT** "assignment of objects of this class to a name" (there's no "assignment TO objects", only OF objects)



- Python raises an exception for errors, e.g.: x=[1,2,3]; x[3]=99 Traceback (most recent call last): IndexError: list assignment index out of range
- You can define your own exception classes: class MyError(Exception): pass
- You can raise any exception in your code: raise NameError, 'unknown name %s' % nn raise MyError, 223961
- You can re-raise the exception last caught: raise

exception handling

```
try:
    x[n] = value
except IndexError:
    x.extend((n-len(x))*[Nonel)
    x.append(value)
else:
    print "assignment succeeded"
f = open('somefile')
try: process(f)
finally: f.close()
```

iterator

iterators may be non-terminating

```
class fibonacci:
    def ___init___(self): self.i=self.j=1
    def __iter__(self): return self
    def next(self):
        r. self.i = self.i, self.j
        self.j += r
        return r
for rabbits in fibonacci():
    if rabbits>100: break
    print rabbits,
1 1 2 3 5 8 13 21 34 55 89
```

iterators can terminate

```
class fibonacci:
    def __init__(self, bound):
        self.i=self.j=1
        self.bound= bound
    def ___iter__(self):
        return self
    def next(self):
        r, self.i = self.i, self.j
        sélf.j += r
if r >= bound: raise StopIteration
        return r
for rabbits in fibonacci(100):
    print rabbits,
1 1 2 3 5 8 13 21 34 55 89
```

generators return iterators

```
from ___future__ import generators # 2.2
def fibonacci(bound):
    r, i, j = 0, 1, 1
    while r < bound:
        if r: yield r
        r, i, j = i, j, j+j
for rabbits in fibonacci(100):
    print rabbits,
1 1 2 3 5 8 13 21 34 55 89
```

generator

generator example: enumerate

```
# it's a built-in in 2.3, but, in 2.2...:
from __future__ import generators
def enumerate(iterable):
    n = 0
    for item in iterable:
        yield n, item
        n += 1
print list(enumerate('ciao'))
[(0, 'c'), (1, 'i'), (2, 'a'), (3, 'o')]
```

importing modules

```
print math.atan2(1,3)
0.321750554397
print atan2(1,3)
Traceback (most recent call last):
NameError: name 'atan2' is not defined
atan2 = math.atan2
print atan2(1,3)
0.321750554397
# or, as a shortcut: from math import atan2
```

import math

standard library module



Even easier...:

- every Python source file wot.py is a module
- can be imported via import wot
 - ...as long as it resides on the import-path
 - ...which is list path in standard module sys
 - sys.path.append('/some/extra/path')
- a module's attributes are its top-level names
 - AKA "global variables" of the module
 - functions and classes are "variables" too



- a package is a module containing other modules
 - possibly including other packages, recursively
- lives in a *directory* containing ___*init__.py*
 - ___*init__.py* is the package's body, may be empty
 - modules in the package are files in the directory
 - sub-packages are sub-directories with ___init__.py
- parent directory must be in sys.path
- imported and used with dot.notation: import email.MIMEImage or: from email import MIMEImage



standard Python library (round numbers...):

- 180 plain modules
 - math, sys, os, sets, struct, re, random, pydoc, gzip, threading...
 - socket, select, urllib, ftplib, rfc822, SimpleXMLRPCServer, ...
- 8 packages with 70 more modules
 - bsddb, compiler, curses, distutils, email, hotshot, logging, xml
- 80 encodings modules
- 280 unit-test modules
- 180 modules in Demo/
- 180 modules in Tools/ (12 major tools+60 minor)
 - compiler, faqwiz, framer, i18n, idle, webchecker, world...
 - byext, classfix, crlf, dutree, mkreal, pindent, ptabs, tabnanny...
- -- but wait! There's more...



other batteries -- GUI and DB

GUIs

- Tkinter (uses Tcl/Tk)
- wxPython (uses wxWindows)
- PyQt (uses Qt)
- Pythonwin (uses MFC -- Windows-only)
- AWT and Swing (Jython-only)
- PyGTK, PyUI, anygui, fltk, FxPy, EasyGUI, ...

DBs (with SQL)

- Gadfly, PySQLite, MkSQL (uses MetaKit)
- MySQL, PostgreSQL, Oracle, DB2, SAP/DB, Firebird...
- JDBC (Jython-only)

other batteries -- computation

- Numeric (and numarray)
- PIL (image processing)
- SciPy
 - weave (inline, blitz, ext_tools)
 - fft, ga, special, integrate, interpolate, optimize, signal, stats...
 - plotting: plt, xplt, gplt, chaco
- gmpy (multi-precision arithmetic, uses GMP)
- pycrypto



- integration with Apache:
 - mod_python
 - PyApache
- high-level packages:
 - WebWare
 - Quixote
- stand-alone (async, highly-scalable):
 - Medusa
 - Twisted





- development environments:
 - Free: IDLE, PythonWin, BOA Constructor, ...
 - Commercial: WingIDE, BlackAdder, PythonWorks, ...
- (GUI builders, debuggers, profilers, ...)
- packagers:
 - distutils, py2exe
 - McMillan's Installer

integration with C/C++/...

extending:

- Python C API
- SWIG
- Boost Python
- CXX, SCXX, sip, ...
- Pyfort, pyrex, ...
- COM (Windows-only), XPCOM, Corba, ...

embedding

- Python C API
- Boost Python (rsn...)



extending:

- transparent: Jython can import Java-coded classes
- from standard libraries, your own, third-party...

embedding

- Jython can implement Java-coded interfaces
- jythonc generates JVM bytecode
- the Jython interpreter is accessible from Java