1. General Python presentation :
    - Why (not) Python ?
    - *History & Community*
    - *Implementations & bundles (flavors)*
    - *Frameworks & libs*
2. Some coding bases
3. Peps & the zen of Python

3

2

1

Why (not) Python

Why not ?

- You need realtime (< µsec)

    but check Viper (http://doc.viperize.it)

- You have very little RAM

    but Tiny Python = 64K & VM of micropython = 2,4K

- You have very little CPU

    but there are implementations even for Arduino

- Compiled code is your God, Interpreted code is your Evil.

- You need protected code

- You need compatibility with bullshit managers

    (justification + job-protection) > technical rationality

Why (not) Python

Why ?

- Very high-level, short yet very readable, intuitive syntax
  - ✓ Easy to learn (pseudo-code-like) *(no funky WTF characters series...follow my eyes)*.
  - ✓ Great for non-trivial business-logic *(You still understand while your client is long time lost in his own specs)*
  - ✓ Maintainable even after 150 last-minute changes *(Don't tell me it never happens)*
  - ✓ Maintainable even after 10,000 lines of code *(Maintenance = €)*
- Totally Free & Truly Open Source (PEPs, PSF, ...)
- Portable on most OS, including embed systems, including GUI lib
- Many different uses:
  - Small helper-scripts *(Any Sys-admins in the room ?)*
  - Softs with GUI *(Cool to build cross-OS heavy clients)*
  - Serious daemons *(numerous frameworks & protocol libs, API-Hour being the best)*
  - Scripting in other softs *(GNU Radio, Blender, OOo, AutoDesk Maya, ArcGIS, GIMP, Minecraft, Corel, Rhino 3D ...)*
  - Live data processing in shell *(Any mad scientist or mathematician in the room ?)*
  - Full engine embed in SSC *(SSCs invasion won't stop anytime soon)*
  - Simplified engine in some micro-controllers
- Huge community with many different types of users

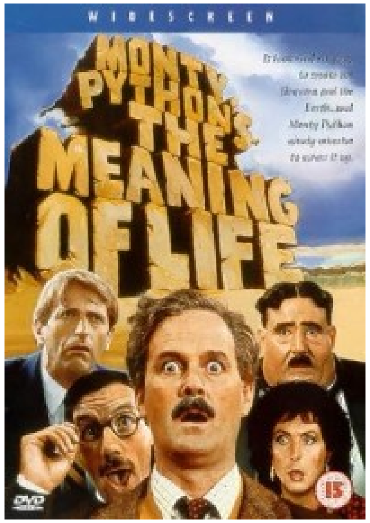
Why (not) Python

- Interpreted with
    - Bytecode caching (.py to .pyc)
    - Interpreter available a runtime *(reason why increasingly used as macro language in apps)*
    - Interactive mode *(Cool for micro testing while coding, cool for quick data processing)*
- Procedural and/or OO *(community diversity => Python is more secular)*
  - multiple inheritance
  - overloading of everything, reflective meta-programming.
  - Introspection *(question, change objects, setters & getters runtime, fn decorators...)*
  - Powerful exceptions handling *(with readable stack-traces...follow my eyes)*
- Strongly typed, non-declarative ("Duck typing"). *(but change is around the corner...)*

History & community:

- Conceived in the 1980's, and implemented by 1990 by Guido van Rossum, named after the BBC show “Monty Python's flying circus”
- 1994: Python 1.0
- 2000: Python 2.0 (true Open Source, community-based project)
- 2001: Python Software Foundation non-profit org
- 2002: Award from the Free Software Foundation
- 2006: Python 2.5 (in all older Unixes)
- 2008: Python 2.6 (in all recent Unixes)
- 2009: Python 3.0
- 2010: Python 2.7 (last of 2)
- 2014: Python 3.4

Belgian beer



python

History & community:

Some of the brands using it for serious and less serious stuff...

Dropbox
reddit
Google
O'REILLY
PHILIPS
BitTorrent
YAHOO! GROUPS
D-Link
Sid Meier's CIVILIZATION IV
rackspace
the #1 managed cloud company
LUCASFILM Ltd
NASA
odoo
(OpenERP)
KNMP
eDF
AstraZeneca
BizRate
HITMEISTER
cpacket NETWORKS
ActiveState
Dynamic Tools for Dynamic Languages
CCP
ARRAY BIOPHARMA
OPSWARE INC
Automating IT
CANONICAL
IRONPORT SYSTEMS
Disney's TOONTOWN ONLINE
ForecastWatch
Accuracy Defined
STi
exoweb
agile software development
ENTHOUGHT

History & community:

- Python Software Foundation non-profit org, many sponsors
- Many annual conferences & workshops:

  DjangoCon Europe
  - EuroPython
  - EuroSciPy
  - Kiwi PyCon (New Zealand)
  - OSCON/IPC (O'Reilly Open Source Convention / International Python Conference)
  - pyArkansas
  - PyCon (original conference, US / North America)
  - PyCon Argentina (formerly Python en Santa Fe (Argentina))
  - PyCon Asia Pacific
  - PyCon AU (Australia)
  - PyCon Brasil
  - PyCon FI (Finland)
  - PyCon FR (France) - Journées Python
  - PyCon DE (Germany)
  - PyCon India
  - PyCon Ireland
  - PyCon Italia (Report on 2007 Conference)
  - PyCon PH (Philippines)
  - PyCon PL (Poland)
  - PyCon UK
  - PyCon Ukraine
  - PyCon ZA (South Africa)
  - PyData
  - PyGotham
  - PyOhio
  - RuPy
  - SciPy (US)
  - SciPy (India)
  - UKUUG Spring Conference 2008 includes a one day Python tutorial
  - Workshop: Python in the German-Speaking Countries

Implementations & Bundles :

Implementations:

- Python (=Cpython)
- PyPy (Interpreter and JIT compiler in Python; !! RPython !!)
- Rpython
- Jython (Java, can extend java classes, compile to java bytecode for perf...)
- IronPython (written in; C# runs on .NET)
- Stackless Python (= Cpython + tasklets & messaging channels)
- Cython / Pyrex (Near Python language for C compiled Python extensions)
- Pyjamas (= Pyjs) Python to HTML + JS compiler + Ajax & UI framework)
- CLPython (Lisp impl.)

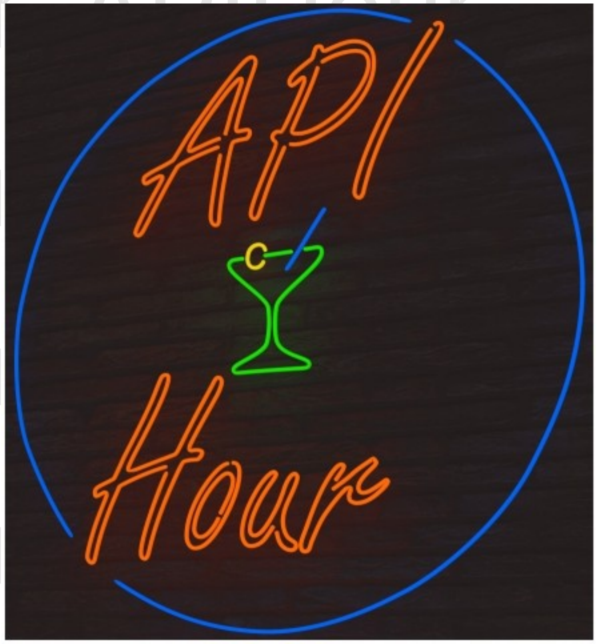
Implementations & Bundles :

Bundles:

- PythonXY (= Python(x,y) = Python SCIentific = Cpython + scientific packages)
- WinPython
- Anaconda
- Python Anywhere

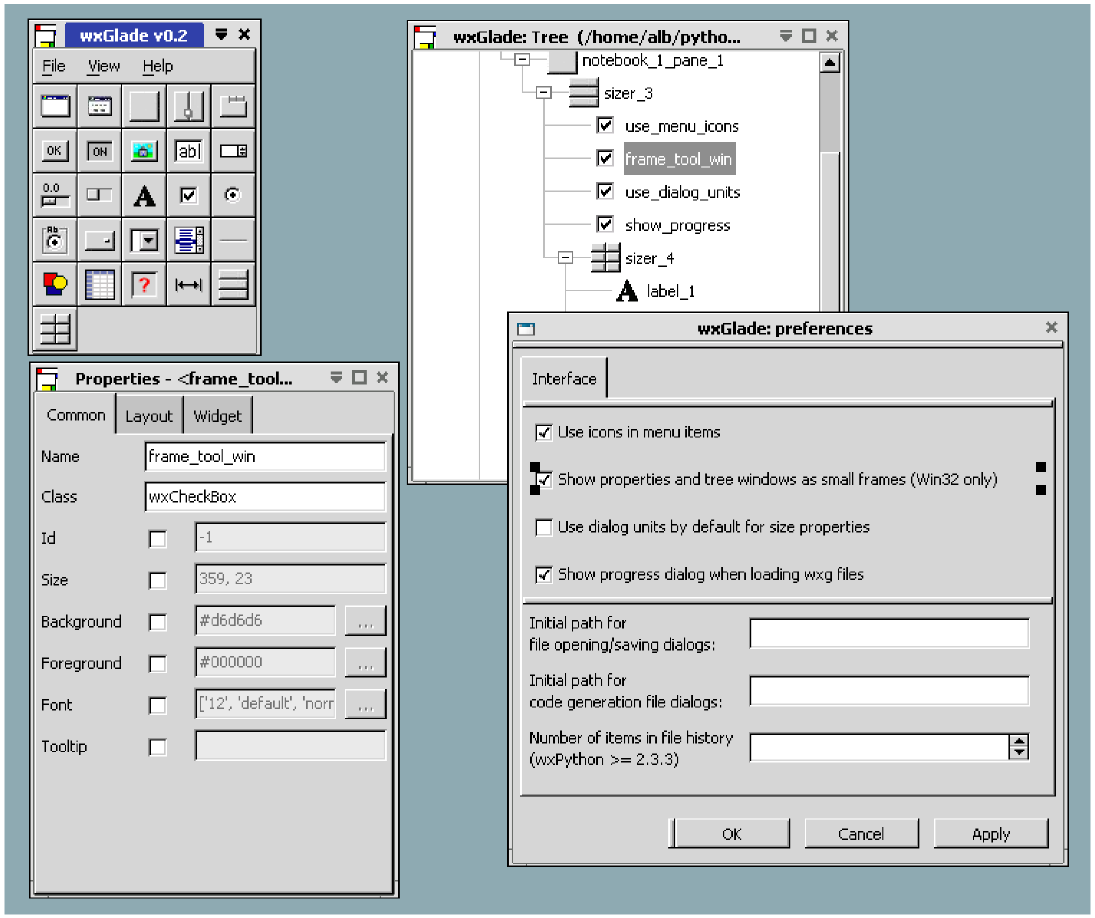
(Web) frameworks:

- APIHour
- Django
- Flask
- Bottle
- Pyramid
- CherryPy
- Pylons
- Zope
- Tornado
- Twisted
- Web2py



GUI libraries:

- TkInter [std lib] (Visual python / Page / PyhtonWorks / Komodo)
- WxPython (Wxglade)
- GnomePython
- PyGtk (=> PyGObject)
- PyQt (Qt Designer)
- PyKDE
- PyGUI
- PyGtk2Tk
- Kivy (Cross platform, multi-touch...)
- ...



[✓] use_menu_icons
[✓] frame_tool_win
[✓] use_dialog_units
[✓] show_progress

Name		frame_tool_win
Class		wxCheckBox
Id	☐	-1
Size	☐	359, 23
Background	☐	#d6d6d6
Foreground	☐	#000000
Font	☐	['12', 'default', 'norr
Tooltip	☐	

[✓] Use icons in menu items
[✓] Show properties and tree windows as small frames (Win32 only)
☐ Use dialog units by default for size properties
[✓] Show progress dialog when loading wxg files

Initial path for file opening/saving dialogs:
Initial path for code generation file dialogs:
Number of items in file history (wxPython >= 2.3.3)

Standard libs:

Numeric and Mathematical Modules
numbers, math, cmath, decimal, fractions, random, statistics,

Functional Programming Modules
itertools, functools, operator

File and Directory Access
pathlib, os.path, fileinput, stat, filecmp, tempfile, glob, fnmatch,
linecache, shutil, macpath

Data Persistence
pickle, copyreg, shelve, marshal, dbm, sqlite3

Data Compression , Archiving and Crypto
zlib, gzip, bz2, lzma, zipfile, tarfile, hashlib, hmac

File Formats
csv, configparser, netrc, xdrlib, plistlib

Generic Operating System Services
os, io, time, argparse, getopt, logging, logging.config, logging.handlers, getpass,
curses, curses.textpad, curses.ascii, curses.panel, platform, errno, ctypes

Concurrent Execution
threading, multiprocessing, concurrent.futures, subprocess, sched, queue,
dummy_threading, _thread, _dummy_thread

Interprocess Communication and Networking
socket, ssl, select, selectors, asyncio , event loop, coroutines and tasks
asyncore, asynchat, signal, mmap

Internet Data Handling
email, json, mailcap, mailbox, mimetypes, base64, binhex, binascii, quopri, uu

Structured Markup Processing Tools
html, html.parser, html.entities

XML Processing Modules
xml.etree.ElementTree, xml.dom, xml.dom.minidom, xml.dom.pulldom, xml.sax,
xml.sax.handler, xml.sax.saxutils, xml.sax.xmlreader, xml.parsers.expat

Internet Protocols and Support
webbrowser, cgi, cgitb, wsgiref, urllib, urllib.request, urllib.response,
urllib.parse, urllib.error, urllib.robotparser, http, http.client, ftplib, poplib,
imaplib, nntplib, smtplib, smtpd, telnetlib, uuid, socketserver, http.server,
http.cookies, http.cookiejar, xmlrpc, xmlrpc.client, xmlrpc.server, ipaddress

Multimedia Services
audioop, aifc, sunau, wave, chunk, colorsys, imghdr, sndhdr, ossaudiodev

Internationalization
gettext, locale

Graphical User Interfaces with Tk
tkinter, tkinter.ttk, tkinter.tix, tkinter.scrolledtext, IDLE

Development Tools
pydoc, doctest, unittest, unittest.mock, unittest.mock, 2to3
test, test.support

Debugging and Profiling
bdb, faulthandler, pdb, timeit, trace, tracemalloc

Software Packaging and Distribution
distutils, ensurepip, venv

Python Runtime Services
sys, sysconfig, builtins, __main__, warnings, contextlib, abc,
atexit, traceback, __future__, gc, inspect, site, fpectl

Custom Python Interpreters
code, codeop,31. Importing Modules
zipimport, pkgutil, modulefinder, runpy, importlib

Python Language Services
parser, ast, symtable, symbol, token, keyword, tokenize, tabnanny,
pyclbr, py_compile, compileall, dis, pickletools

Misc / Windows / Unix
Formatter / msilib, msvcrt, winreg, winsound / posix, pwd, spwd, grp, crypt, termios, tty,
pty, fcntl, pipes, resource, nis, syslog

Non standard libs:

Pypi: The Python package index repository

- About **65.000** Packages !
- Super easy install : “pip install xxx”
- Avoid system pollution: Virtual-Env

matplotlib

SciPy.org

PyOpenGL

Python-MySQL

PyUSB

NumPy

Python
Imaging
Library

Python-PostgreSQL

PySERIAL

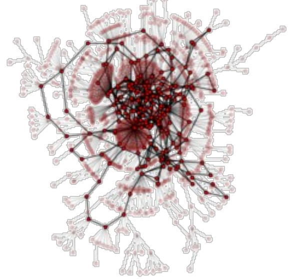
OpenCV Python

pySonic

pygame

PYTHON-OGRE

NetworkX



pyro

PySoy 3D game engine

SymPy

VPYTHON

Quick search on Github for Python Ham-related:

- •APRSlib
- •QUISK (SDR)
- •HamRadioLogbook
- •SkHamPy
- •N1YWB Python Ham Radio Tools :
  geolog, hamtools.adif, hamtools.qrz,
  hamtools.ctydat, hamtools.kml, VoiceKeyer
- •WxContestLogger
- •DX-Cluster-Parser
- •PyMorse
- •PyKeyer (Reec)
- • ...

Coding:

Python peculiarities

- **Indentation = 2spaces or 4 spaces = ECMA curly brackets**

```
if(x==5):
    print(“That's a five!”)
else:
    print(“That's not a five !”)
print('Out of test...')
```

Zen of Python says:
Keep your lines short

- **use “\” for multi-line statements (not for inline lists or dicts)**

```
big_total= first_thing + \
                  second_thing

x = ['aaa','bbb','ccc',
       'ddd','eee']
```

- **use ””” for long multi-line literals**

```
big_string=”””Lorem ipsum dolor sit amet, consectetur adipiscing elit.
                        Nunc turpis nunc, dignissim vitae luctus ac, porttitor tempus
                        magna. Mauris ultrices dui ante, a facilisis leo. Duis nec mi eu sem
                        blandit sodales id vitae massa. ”””
```

- **use # for comments (””” xyz ””” for docstrings)**

- **No ++ or --, but +=   -=   *=   /=**

Coding:

Syntax overview

```
if( (x==1) and (y!=2)):
    [block]
elif:
    [block]
else:
    [block]
```

```
while():
    [block]
    if(x==1): break
    if(x==2): continue
    [block]
```

```
for item in sequence_item:  #(iterable)
    [block]

for item in generator: #(iterator)
    [block]

for x in range(1,10):  #(iterable)
  print x
```

```
try:
    [block_to_try]
except ZeroDivisionError:
    [block]
except ValueError:
    [block]
else
    [block (only if block_to_try went ok)]
finally
    [always executed on way out]

###################################
raise ZeroDivisionError
```

```
class MyClass(MyParent):

    def __init__(self):
       self.property=”toto”

    def method1(self,param='default'):
       [block]
       return(x)
```

Coding:

Python types (1/2)

Numbers:

int:	x1=5	**!!! x/3 = 1 !!!**
long:	x2=45L ; x2=long(45)	
float:	x3=45.23 ; x4=6.42e3 ; x5=6.43e-5	
complex:	x6=1.23+4.5j ; x6=complex(1.23,4.5)	

Casting:

```
int(x3)       #45
float(x1/3)  #1.0
float(x1)/3  #1.6666666666666667
```

Ex. of methods & attributes of these objects:

```
x3.is_integer()    # False
x7.imag            #4,5
```

Other bases:

x=int('F0',16) ; x=0xF0 ; x=0b11110000

Special “emptyness” type:

x=None

!!! different than '', [], {}, 0, False !!!

Booleans:

```
b=True      # b*5=5  and type(b) is still 'bool'
b=False     # b*5=0  and type(b) is still 'bool'
```

Iterable types (Sequences):

strings, unicode, list, tuple, bytearray, buffer, xrange

-more in next slide-

Iterators types (Generators):

An iterator type object implements **next()** method, which raises a **StopIteration** exception when called on last iteration,

File is a special (enriched) iterator (no other built-in type)

Mapping types (hashes): dict

```
d={'first':'me','second':222,'third':False}
d['fourth']=45.67
print(d['second'])    #222
```

Dictionaries methods:

clear(), copy(), fromkeys(kseq,v), get(k, default), has_key(k), items(), iteritems(), iterkeys(), itervalues(), keys(), values(), pop(key,default), popitem(), update(d2)

Coding:

Python types (2/2)

Strings:

```
s1='coucou' ; s="coucou"

s2="""very long way of
saying coucou"""

s3='hello %s %i %1.1f' %(t,x,f)
```

Many usual string methods

Because a string is a sequence:

```
print(s2[3])   #y#
print(s2[0:4])   #very#
print(s2[5:])   #long way of saying coucou#
print(s2[5:-6])   #long way of saying #
print(s2[:-6].upper())   #VERY LONG WAY OF SAYING #
print(s2[-13:].replace('cou','to'))   #saying toto#
print(s2[3]*5]  #yyyyy"

for letter in s2:
  print(letter)

if('bidule' in s2):
```

Lists:

```
li=['aa',12,'bb',23e-2,'cc']
```

Lists methods:
append(elem), extend(lst), count(elem), index(elem,start,end), insert(i,elem), pop(i), remove(elem), reverse(),sort(cmp,k,reverse))

Coding:

Python modules & packages

Importing modules:

```
### Basic import:
import mylib
obj=mylib.myclass1()
result=mylib.staticfunction(param)

### Basic import with namespace change:
import mylib as ml
obj=ml.myclass1()
result=ml.staticfunction(param)

### Selective import with namespace merging :
from mylib import myclass1
obj=myclass1()

### import everything with namespace merging :
from mylib import *
obj=myclass1()
result=staticfunction(param)
```

Importing packages:

```
### General
from mypkg import *

### Selective
from mypkg import module1,subdir.module2
```

Modules:

File: mylid.py

```
class myclass1():
 ###class definition ...

class myclass2():
 ###class definition …
```

Packages:

Use a __**init__.py** file,

that defines
__all__=['module1','module2']

and eventually
__path__=['mypkg','mypkg/subdir']

Special case:

```
from __future__ import *
```

Coding:

Class specialties

Special methods :

__new__ : static, called to create instance
__init__ : usual constructor (instance created)
__del__ : usual destructor

__repr__: python (debug) string representation
__str__: called by str() & print

__lt__, __le__, __eq__, __ne__,
__gt__, __ge__, __cmp__: comparison operators

__getattr__: called on attribute lookup & not found
__getattribute__: called unconditionally on attribute lookup
__setattr__: called on attribute assignment
__delattr__: called on attribute deletion

__len__: len()
(and several other methods for container type objects)

(methods for sequence type objects)

(methods for numeric type objects)

Special attributes :

__doc__ : docstring
__name__ : current function or class name
__dict__ : objects writable attributes
__bases__: tuple of class parents
also built-in function **dir()**

Typical:

```
if(__name__ == "__main__"):
  [Do the main thing]
```

Special built-in functions :

callable(obj)
dir(obj)
delattr(obj,'attrname') # same as del(obj.attrname)
hasattr(obj,'attrname')
help(obj) # interactive mode
id(obj)
isinstance(obj,classinfo)
setattr(obj,'attrname',v)
str(obj)
super()
type(obj)

Peps & the zen of Python:

What's a pep ?

See PEP0 ;-)
Python Enhancement Proposal

Good coding conventions: PEP8

- Indentation: Use 4 spaces per indentation level.

- Don't use ';'

- Avoid very long lines (PEP8 says 79, I say < 140)

- Use docstrings

- Put spaces around == , < , > , != , <> , <= , >=

- Naming :

  Package and Module Names: lowercase

  Class Names: Camelcase

  Function or method Names: lowercase

  Variables: lowercase (with _)

Peps & the zen of Python:

- Beautiful is better than ugly.
- Explicit is better than implicit.
- Simple is better than complex.
- Complex is better than complicated.
- Flat is better than nested.
- Sparse is better than dense.
- Readability counts.
- Special cases aren't special enough to break the rules.
- Although practicality beats purity.
- Errors should never pass silently.
- Unless explicitly silenced.
- In the face of ambiguity, refuse the temptation to guess.
- There should be one-- and preferably only one --obvious way to do it.
- Although that way may not be obvious at first unless you're Dutch.
- Now is better than never.
- Although never is often better than *right* now.
- If the implementation is hard to explain, it's a bad idea.
- If the implementation is easy to explain, it may be a good idea.
- Namespaces are one honking great idea -- let's do more of those!