Basics

Enough background to parse files

---

by

David Roubinet , Jan14

Presentation powered by reveal.js

home

What is Python ?

- v2.0 released in 2000 by Guido Van Rossum
- Quoting Monty Python is a healthy habit
- Quoting StarWars is borderline, but OK too

Why ?

Easy on the *developer*, not on the machine

Used ?

Google, Youtube, Academics, Labs

For us ?

- Analyze logs, generate plots, reports
- Glue or bridge existing apps, db, servers

Setup

Version

v2.7 recommended

v3.3 nicer syntax but fewer libraries

Install

Linux & Mac : already there

Windows : get it @ python-2.7.6.msi

Configure your text-editor / IDE

no TAB → emulated with 4 spaces

Hello World

Create a file `helloWorld.py`

```
#!/usr/bin/env python
print "Hello World!"
```

Lauch on any os :

```
/<path>/<to>/python helloWorld.py
```

Launch on Mac/Linux :

```
/usr/bin/env python --version # test your shebang
chmod +x ./helloWorld.py      # make executable
./helloWorld.py
```

Interactive Shell

python with *no argument* is a console …

```
Python 2.7.4 (default, Sep 26 2013, 03:20:56)
[GCC 4.7.3] on linux2
Type "help", "copyright", "credits" or "license" for more info.
>>> print 1+2
3
>>> exit()
```

Handy calculator accepting *big* numbers.

```
>>> 123**102
1480203295629928329479981685264795353307343870806030995772572
2520911504293200426656620631029510757221888460901007696324336
8523439832863460122438964502975350810638854406478502135611850
1231433129L
```

```
>>> (9j+2)**2
(-77+36j)
```

```
>>> hex(0x55^0b11111111)
'0xaa'
```

Strings

Quoting

Many delimiters allowed: `'  ''  '''  "  ""  """`

A delimiter encloses another: `'He said "Hello"'`

```
message = """ This is a  very very very very  very very
very very very very very very long multi-line string.
                         Victor Hugo         """
```

Formatting : string % (values)

```
>>>"There are %d %s %s(s)"%(2,"red","banana")
'There are 2 red banana(s)'
```

Formatting : string % { values }

```
>>>"%(boy)s runs fast. %(boy)s is %(age)d"%{"boy":"Bob","age":12}
'Bob runs fast. Bob is 12'
```

String Cont'd

Operations

	Expression	*Result*
concatenate	`'ABC'+'DEF'*2`	`'ABCDEFDEF'`
1st, 2nd	`'ABCDEF'[0],'ABCDEF'[1]`	`('A','B')`
last, before	`'ABCDEF'[-1],'ABCDEF'[-2]`	`('F','E')`
2nd to 3rd	`'ABCDEF'[1:3]`	`'BC'`
2nd to 2nd	`'ABCDEF'[1:2]`	`'B'`
omit *first*	`'ABCDEF'[:2]`	`'AB'`
omit *last*	`'ABCDEF'[-2:]`	`'EF'`
always full	`'ABCDEF'[:i]+'ABCDEF'[i:]`	`'ABCDEF'`

Trick: Think of slicing[i:j] as slicing[i:j[

Quiz

With : `vals=(9,8,7,6,5,4,3,2,1,0)`

display : `'9->8->7->6->5->4->3->2->1->0->bing'`

Answer

```
('%d->'*10+'bing')%vals
'%d->%d->%d->%d->%d->%d->%d->%d->%d->%d->bing'%vals
'9->8->7->6->5->4->3->2->1->0->bing'
```

Lists

Square brackets

```
mylist = ['AB','cd',23,0x34,"EF"]
```

Trailing comma allowed

```
mylist2 = ['AB',
           0x34,
         # "EF", # Commenting out doesn't break syntax
          ]
```

Operations

	Expression	Result
concatenate	`['AB','CD']+['EF']*2`	`['AB','CD','EF','EF']`
indexing	`['AB','CD','EF'][0]`	`'AB'`
slicing	`['AB','CD','EF'][1:3]`	`['CD','EF']`
existence	`'CD' in ['AB','CD','EF']`	`True`

Quiz ?

```
[1,2,3,4,5][2:2]
```

Answer:

- `[1,2,3,4,5][2:2]` = `[]` = empty list
- Same goes for `mylist[i:j]` whenever `i` ≥ `j`

Dictionaries

Curly Braces

```
mydict = { 'price'  : 12,
           'type'   : 'Table',
           'options': ["red","blue"],
           'dim'    : {"W":90,"L":180,"H":72},
            }
```

Operations

	Expression	Result
keyword indexing	`{'A':12,'B':'C'}['A']`	`12`
existence	`'C' in {'A':12,'B':'C'}`	`False`
modify entry	`d={'A':12,'B':'C'};d['A']=13`	`{'A':13,'B':'C'}`
add entry	`d={'A':12,'B':'C'};d['Z']=0`	`{'A':12,'B':'C','Z':0}`
remove entry	`d={'A':12,'B':'C'};del(d['B'])`	`{'A':12}`

Quiz

```
mydict = { 'price'   : 12,
           'type'    : 'Table',
           'options': ["red","blue"],
           'dim'     : {"W":90,"L":180,"H":72},
            }
```

```
mydict['options'][1][-2]

mydict['dim']['H']-mydict['price']
```

Answer

`mydict['options'][1][-2]`

```
mydict['options']        = ["red","blue"]
mydict['options'][1]     =       "blue"
mydict['options'][1][-2] =         "u"
```

`mydict['dim'][H]-mydict['price']`

```
mydict['dim']           = {"W":90,"L":180,"H":72}
mydict['dim']["H"]      = 72
mydict['price']         = 12
                        -----
                        = 60
```

Introspection

Everything is an object

```
>>> dir('ABCDEF')
['__add__', '__class__', '__contains__', '__delattr__',
...
'lower', 'lstrip', 'partition', 'replace', 'rfind',
'strip', 'swapcase', 'title', 'translate', 'upper']
```

Methods

```
>>> 'ABCDEF'.lower
<built-in method lower of str object at 0xb742b8e0>
>>> 'ABCDEF'.lower()
"abcdef"
```

Doc

```
>>> help("ABCDEF".split)
split(...)
    S.split([sep [,maxsplit]]) -> list of strings
```

Quiz

```
msg="These are not the droids you're looking for."
```

Use `dir(msg)`, and find a method to substitute *droids* with *burgers*.

Answer

```
msg.replace("droids","burgers")
```

```
"These are not the burgers you're looking for."
```

Usual methods

	Expression	Result
text trim	`'A BC DEF\n'.strip()`	`'A BC DEF'`
text search	`"ABCDEFABCDEF".find("EF")`	`'4'`
text replace	`'ABCDEFABCDEF'.replace('A','aa')`	`'aaBCDEFaaBCDEF'`
text -> list	`'A BC DEF'.split()`	`['A','BC','DEF']`
list -> text	`'-'.join(['A','BC','DEF'])`	`'A-BC-DEF'`
list sort in place	`l=[6,3,1,2]; l.sort()`	`[1,2,3,6]`
list reverse in place	`l=[6,3,1,2]; l.reverse()`	`[2,1,3,6]`
dict keys	`{'A':12,'B':'4'}.keys()`	`['A','B']`
dict values	`{'A':12,'B':'4'}.values()`	`['12','4']`

Indentation

Indentation is part of the syntax

```
if True:              #    <- First statement is in column 0
    print "L"         # x  <- Choose any indentation
    print "O"         # x  <- Stick with it for current block
    if True:          # x  <- And again
        print "V"     #    <- New block... new choice
                      #    <- Empty line any time
    print "E"         # x  <- Back to first choice
  print "Cheese"      #    <- Syntax Error
```

Any choice is possible, but :

- please avoid tabs, as they depend on editor' configurations
- consider multiple of 4 spaces

Conditions

```
if   color is "yellow"                    : guess="banana"  # simple
elif color=="" and basket=="empty"        : guess="no more" # and
elif color in ["red","green"]             : guess="apple"   # or
elif color is not "orange"                : guess="yogurt"  # not
elif type(color)==type(1) and color>4 : guess="coconut" # shielded
else                                      : guess="orange"
```

```
if 1:  # Easy to comment in/out
   print "debug stuff"
```

```
if A:
  if B: myfunc(1)
  else: myfunc(2)
else:
  if B: myfunc(3)
  else: pass      # Optional, but good to know
```

Iterate

```
>>> for char in "YMCA": print char,
Y M C A

>>> for word in ["I","am","Ironman"]: print word,
I am Ironman

>>> for x in range(10): print x,
0 1 2 3 4 5 6 7 8 9

>>> for man    in phonebook: print man,phonebook[man]
>>> for cow    in field: ...
>>> for child  in classRoom: ...
>>> for widget in gui: ...
```

Don't Count

```
for i in range(0,len(myList)): print myList[i],
```

Quiz

Look at `help()` on `range` and display the 10 first multiples of 7

Quiz

Display the phone book, sorted alphabetically.

```
phonebook = {"Bob":1234,"Alice":3456,"Charly":4567}
```

Answer

```
>>> help(range)
range(...)
    range(stop) -> list of integers
    range(start, stop[, step]) -> list of integers

Return a list containing an arithmetic progression of integers.
range(i, j) returns [i, i+1, i+2, ..., j-1]; start (!) defaults to 0.
When step is given, it specifies the increment (or decrement).
For example, range(4) returns [0, 1, 2, 3]. The end point is omitted!
These are exactly the valid indices for a list of 4 elements.
```

```
>>> range(0,7*10,7)
[0, 7, 14, 21, 28, 35, 42, 49, 56, 63]
```

Answer

```
phonebook = {"Bob":1234,"Alice":3456,"Charly":4567}
names = phonebook.keys()
names.sort()
for name in names:
   print name,phonebook[name]
```

```
Alice 3456
Bob 1234
Charly 4567
```

Functions

```
def myfunc(x,y,verbose=False):
    result = x+y
    if verbose:
        print "Info: %r+%r = %r"%(x,y,result)
    return(result)
myfunc("SAY"," CHEESE",verbose=True)
myfunc(23,17) # verbose is optional
```

```
Info: 'SAY'+' CHEESE' = 'SAY CHEESE'
'SAY CHEESE'
40
```

First class citizen

```
>>> myvar=myfunc
>>> callable(myvar)
True
>>> myvar(1,2)
```

Quiz

Are methods callable ?

Answer

```
>>> callable("ABC".split)
True
```

Parsing

Text files In

```
for line in file("myFile.txt"):
    print "Reading:",line,
```

Text files In/out

```
fo = file("myFileOut.txt","w")
for line in file("myFile.txt"):
    print >> fo, "Reading:",line,
fo.close()
```

XML files

```
import xml.etree.ElementTree as ET
for child in ET.parse("myFile.xml").getroot():
   print child.tag, child.attrib
```

Parsing Cont'd

Web pages

```
import urllib
for line in urllib.urlopen("http://www.google.com"):
    print "Reading:",line,
```

JSON files

```
import json
mydict = json.load("myfile.json")
```

Compressed files

```
import gzip
for line in gzip.open('myfile.txt.gz', 'rb'):
   print line,
```

Modules

Imported only once

```
import math
import math # <- This is skipped
```

File structure

```
import thepack.subpack.thismodule
```

```
thepack/ # searched 1st in "." then $path,$PYTHONPATH,python install
       /__init__.py
       /subpack/
               /__init__.py
               /thismodule.py
```

Run time setup

```
import os,sys
sys.path.insert(0,os.environ["MYLIBDIR"])
import mylib
print mylib.__file__ # Be sure of what is imported
```

Darkside

```
from thismodule import *
from thatmodule import *
...
awesomeFunction() # Where is it from ?
```

Luke : What's in there?

Yoda : What you only with you take.

Embedding doc

```
def greetings(name="You"):
   """
   If the first statement is in fact a string....
   It is considered as a comment, and will be forwarded by
   documentation utilities ( pydoc, epydoc, sphinx )
   """
   print "Hello",name
```

Query

```
>>> help(greetings)
greetings(name='You')
    If the first statement is in fact a string....
    It is considered as a comment, and will be forwarded by
    documentation utilities ( pydoc, epydoc, sphinx )
```

Publish

```
$ python -m pydoc -w greetings # produces greetings.html
```

Embedding test

`mymodule.py`:

```
def func(x,y): return (x+y)

if __name__=="__main__": # <- false if current code is imported
  print "selftest..."
  assert func(1,2)==3
  assert func(4,2)==6
  print "PASS"
```

Usage

```
import mymodule
mymodule.func(4,5) # <- Access via name space
```

Testing

```
$ python mymodule.py
selftest
PASS
```

Command line arguments

```
import sys
```

```
print "Script:",sys.argv[0]

for arg in sys.argv[1:]: print "Argument:",arg

verbose =  "-v" in sys.argv
```

```
>> python thisFile.py A BC D
Script: thisFile.py
Argument: A
Argument: BC
Argument: D
```

Exit Code

Possible	Better	after execution
`sys.exit(0)`	`raise SystemExit`	`echo $status` → 0
`sys.exit(1)`	`raise SystemError`	`echo $status` → 1

File system

```
import os,shutil,glob
```

csh	python
`ls mydir/*.txt`	`glob.glob("mydir/*.txt")`
`if (-e myfile)`	`if os.path.exists("myfile"):`
`if (-d mydir)`	`if os.path.isdir("mydir"):`
`mkdir mydir`	`os.mkdir("mydir")`
`rm -rf mydir`	`shutil.rmtree("mydir")`
`cp $src $dst`	`shutil.copyfile(src,dst)`
`$mypath:t`	`os.path.basename(mypath)`
`$mypath:r`	`os.path.dirname(mypath)`
pwd	`os.getcwd()`

Regular Expressions

```
import re
```

`Hel*o.` : matches Hellllllo!

Usage

Expression	*Result*
`re.search('B.D' ,'ABCDEF')`	True, contains BCD
`re.match( 'AB' ,'ABCDEF')`	True, starts with AB
`re.findall('[AD].','ABCDEF')`	`['AB','DE']`
`re.search('(A.)CD(.)F','ABCDEF').groups()`	`('AB','E')`
`re.sub('A.','ab','ABCABC')`	`'abCabC'`

metacharacters

`.`	Any single character	`^`	Start of line
`$`	End of line	`*`	Repeat 0 or more
`+`	repeat 1 or more	`?`	Repeat 0 or 1
`*?`,`+?`,`??`	Same as `*`, `+`, `?`, but non-greedy	`[abc]`	Any character a, b or c
`{n}`	repeat n times	`{n,m}`,`{,m}`,`{n,}`	repeat range
`[a-d]`	Any character a,b,c or d	`[-bc]`	Any character -,b, or c
`[^abc]`	Any but not a,b nor c	`[a^bc]`	Any character a,^,b or c
`\d`	A digit `[0-9]`	`\D`	Not a digit `[^0-9]`
`\s`	A whitespace `[ \t\n\r\f\v]`	`\S`	Not a white space
`\w`	An alphanumeric `[a-zA-Z0-9_]`	`\W`	Not an alphanumeric
`\A`	Start of string	`\Z`	End of string
`\b`	empty character at border \w ↔ \W	`\B`	empty character not on border

Quiz

Find couples in…

```
msg = "Size1=32,Size3=54,Size12:128; Size7 = 87"
```

Answer

```
re.findall("Size(\d*)[ =:]+(\d*)[,;]*",msg)
```

```
[('1', '32'), ('3', '54'), ('12', '128'), ('7', '87')]
```

Conversions

`int("1010"),int("1010",16),int("1010",2)`	`1010,4112,10`
`list("AB12")`	`['A','B','1','2']`
`eval("3*5+1")`	`16`
`str(123)`	`'123'`
`'%04X'%(252)`	`'00FC'`
`'25/12/2014'.split('/')`	`['25','12','2014']`
`'-'.join(['25','12','2014'])`	`'25-12-2014'`
`ord('a')`	`97`
`'ab^B'.encode('hex')`	`61625e42`

Quiz

Convert `"You're the Doc, Doc"` into a list of ascii-code bytes

Answer

```
mylist=[]
for x in "You're the Doc, Doc":
   mylist.append(ord(x))
```

```
[89, 111, 117, 39, 114, 101, 32, 116, 104, 101, 32, 68, 111,

99, 44, 32, 68, 111, 99]
```

Or

```
[ord(x) for x in "You're the Doc, Doc"]
```

Lab

Given the non-regression report below:

→ list the failing tests

→ report the percentage of success per suite

```
inputData = """\
Suite,      Test,    Status
v2,         mini,    Pass
v2,         mini2,   Pass
Legacy,     basic,   Pass
v2,         mini3,   Pass
Local,      test2,   Pass
v2,         full,    Pass
Local,      test1,   Pass
Legacy,     extra,   Fail
Local,      test3,   Fail
Blind,      ztest,   Fail
Local,      test4,   Pass""".split("\n")
```

- Ready to look at ↓ ↓ solution ↓ ↓

```
db    = {}
fails = []
for line in inputData[1:]: # <- Skipping header row
    suite,test,status = line.replace(" ","").split(",")
    if suite not in db: db[suite]={"totalCount":0,"passCount":0}
    db[suite]["totalCount"] += 1
    if status=="Pass": db[suite]["passCount"] += 1
    else             : fails.append("%s -> %s"%(suite,test))

for fail in fails: print "FAIL:",fail

for suite in db:
    print "STAT:",suite,100*db[suite]["passCount"]/db[suite]["totalCount"]
```

```
FAIL: Legacy -> extra
FAIL: Local -> test3
FAIL: Blind -> ztest
STAT: Blind 0
STAT: v2 100
STAT: Legacy 50
STAT: Local 75
```