Introduction to Python

Python is easy

Python

Disclaimer: Python and Linux fan ! :)

Trivia

Open source, created by Guido van Rossum in 1989/1991.
Intended as a general purpose programming language to be taught to non-computer science students!
Named after Monty Python comedy series.

#1 programming language in Linux Journal's Reader's Choice for 2013 !
~#4 programming language as source code base according to various indexes

It's implemented everywhere: Solaris, Android (SL4A/QPython), (older: Symbian, MS-DOS, PalmOS)
Companies that use it: NASA, Google, Industrial Light & Magic, Rackspace, Dropbox, Spotify
Used in these web frameworks: Django, Zope
Used in these apps: YouTube and DropBox, Reddit, Quora
Python bindings: OpenGL, QT, GTK,

Unix/CLI

Why Unix/CLI (Command Line Interface) ?

- easier tool chaining (The whole is greater than the sum of its parts)
- scripting/automating
- uses less resources
- easier remote access

Why data in text format ?

- no proprietary (binary) format
- in some respect no interface needed between communicating processes
- easy to reverse engineer/parse
- easy to debug

Unix/CLI

Command line arguments

```
> executable [argv1 argv2 argv3 …]
```

Piping/chaining

```
> tool1 | tool2 [ | tool3 …]
```

Streams

```
stdin: standard input (eg: keyboard)
stdout: standard output (eg: printed out somewhere on screen)
stderr: standard error (by default printed out where stdout is printed)
```

Unix/CLI

Streams/redirecting

```
(This is the bash shell syntax:)

cat input.txt | tool1 | tool2 > output.txt
mytool input.txt > output.txt
mytool input.txt > output.txt 2> errors.txt
mytool input.txt 2> /dev/null
```

Unix/CLI

Piping/chaining

```
USER        PID %CPU %MEM    VSZ   RSS TTY        STAT START   TIME COMMAND
dan        7983  0.0  0.1 235596 16852 ?          Ss   06:58   0:02 vim -name vim -
dan        8805  0.0  0.1 235012 16320 ?          Ss   08:03   0:00 vim -name notes
dan        9193  0.0  0.1 232744 15648 ?          Ss   08:31   0:00 vim -name vim -
dan        9603  0.0  0.1 232848 15840 ?          Ss   08:48   0:00 vim -name vim -

> ps axuw | grep dan | grep vim | grep -v grep | awk '{print $5}' | add
235596
235012
232744
232848
936200.0
```

Unix/CLI

Where Unix ?

Linux: is obviously Unix

Mac: is Unix (FreeBSD)

Windows: Cygwin

Android: terminal/Linux

How to run python on Unix ?

- interactive shell

- python script.py

- hasbang line (first line of a script):

```
#!/usr/bin/env python3
```

Git

Version control system

Why is it needed:
- easily see what the changes are
- trackable history
- easy roll back
- distributed copy (backup)
- easy sharing

```
git clone https://github.
com/dnastase/pylec
git add grep.py
git commit [grep.py] -m "created it"
git push
```

```
… make changes

git diff
git commit[grep.py] -m "added case
insensitive"
git push

git pull
```

Python

Language traits

general-purpose

high level

established, mature language

multi-platform: Linux, Mac, Windows, Android
multi-paradigm: structured programming, object oriented, functional programming, meta programming
interpreted language
dynamic language
automatic memory management
simple and consistent syntax
big standard library
interactive shell allows for quick tests and discovery
allows rapid prototyping; building the solution from simple to complex while continuously testing

Python: Readable

```python
import sys

string = sys.argv[1]
inputFile = sys.argv[2]

for line in open(inputFile).readlines():
   if string in line:
      print(line, end='')
```

Python: interactive shell

```
dan@debpc~>
python
Python 2.7.6 (default, Dec 30 2013, 14:37:40)
[GCC 4.8.2] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello world")
Hello world
>>>
```

Python: Syntax

Indentation matters!

```
string = args[1]
inputFile = args[2]

for line in open(inputFile).readlines():
    line = line.rstrip("\n")

    if not options.ignore_case and string in line:
        print(line)
        LOG("matched with case %s in line %s" % (string, line))
    elif options.ignore_case and string.lower() in line.lower():
        print(line)
        LOG("matched case insensitive %s in line %s" % (string, line))
    else:
        LOG("no match for %s in line %s" % (string, line))
```

Python: syntax

Other than at the begining, spaces don't matter.

Newlines don't matter either.

Comments: from # until the end of line

```
#parse the command line arguments
(options, args) = cmdLineParser.parse_args(sys.argv)

string = args[1]        #this is the string to search for
inputFile = args[2]     #this is the file to search into
```

Python: Constants, Variables

```
True, False
2, 3.14
'string', "string", r'string\n'
"""this is
a multi-line
string"""


>>> print("nr", integer_nr)
nr 2
>>> print("nr %03d" % integer_nr)
nr 002
```

Python: Assignment/Creation

```
variable = expression


integer_nr = 2
rationalNr = 3.14
str1 = ‘string’
str2 = “string”
str3 = r’string\n’
str4 = “””this is
a multi-line
string”””

res = myFunction()
obj = MyClass()
```

Python: operators

The usual mathematical, relational and logical operators.

+ - * / % **

== != <> > < >=<= & | ^ ~ << >>

and or not

+= -= *= /= %=

Plus:

- **in**: if element is in sequence
- **not in**
- **is**: checks identity (id() builtin function)
- **is not**

Python: If

```
if condition1:
    body1
elif condition2:
    body2
elif condition3:
    body3
…
else:
    bodyN
```

```
if not options.ignore_case and string in line:
    print(line)
    LOG("matched with case %s in line %s" % (string, line))
elif options.ignore_case and string.lower() in line.lower():
    print(line)
    LOG("matched w/out case %s in line %s" % (string, line))
else:
    LOG("no match for %s in line %s" % (string, line))
```

Python: Lists

Creating

```
>>> l = []
>>> l = [2, 3.14, "py", n, f, s]
>>> l
[2, 3.14, 'py', 2, 3.14, 'string']
>>> l3 = list(open(“file.txt”).readlines())
>>> l2 = list(“abc”)
>>> l2
['a', 'b', 'c']
```

Accessing

```
>>> l[1]
3.14
>>> l[1:3]
[3.14, 'py']
>>> l[-1]
'string'
l[2:]
l[:]
>>> len(l)
5
```

Python: Lists

Adding

```
>>> l
[2, 3.14, 'py', 2, 3.14, 'string']

>>> l.append("new")
>>> l
[2, 3.14, 'py', 2, 3.14, 'string','new']

>>> l.insert(1, 44)
>>> l
[2, 44, 3.14, 2, 3.14, 'string', 'new']

>>> l[1:1] = [55]
>>> l
[2, 55, 44, 3.14, 2, 3.14, 'string',
'new']
```

Deleting

```
>>> del l[1]
>>> l
[2, 44, 3.14, 2, 3.14, 'string', 'new']

>>> l.pop(1)
44
>>> l
[2, 3.14, 2, 3.14, 'string', 'new']

>>> l.remove("new")
>>> l
[2, 3.14, 2, 3.14, 'string']
```

Python: Lists

Modify

```
>>> l=[7, 6, 8, 3.14]

>>> l[0:2]=[3, 1, 2]
>>> l
[3, 1, 2, 8, 3.14]

>>> l.sort()
>>> l
[1, 2, 3, 3.14, 8]
```

Search

```
>>> l.index(3.14)
3
```

Iterating

```
for val in l:
   pass
```

Python: Loops

```
for item in sequence:
     body
else:
     body

while condition:
     body
else:
     body
```

break: exit the loop

continue: skip remainder of the body

else: execute when normally exiting the loop (eg: not **break**)

```
for f in sys.argv[2:]:
   for line in open(f):
      if sys.argv[1] in line:
         found = True
         break
   else:
      found = False
   if not found:
      continue
   print("File %s has '%s'" % (f, sys.
argv[1]))
```

Python: Loops

```
done = False
while not done:
    cmd = raw_input("new command:")
    if 'exit' in cmd:
        done = True
    elif 'show' in cmd:
        DoShow()
    elif 'assign' in cmd:
        DoAssign()
```

Python: Dictionaries

Creating

```
>>> d={'john': 12, 'mary': 34, 'brad':56}
>>> d
{'brad': 56, 'mary': 34, 'john': 12}
>>> d=dict(john=12, mary=34, brad=56)
>>> d
{'brad': 56, 'mary': 34, 'john': 12}
```

Accessing

```
>>> d['mary']
34
>>> d.keys()
dict_keys(['brad', 'mary', 'john'])
>>> d.values()
dict_values([56, 34, 12])
>>> d.items()
dict_items([('brad', 56), ('mary', 34),
('john', 12)])
```

Python: Dictionaries

Adding/Modifying

```
>>> d['ana']=78
>>> d
{'brad': 56, 'mary': 34, 'john': 12,
'ana': 78}

>>> d.update({'steve':90})
>>> d
{'brad': 56, 'mary': 34, 'steve': 90,
'john': 12, 'ana': 78}
```

Deleting

```
>>> del d['ana']
>>> d
{'brad': 56, 'mary': 34, 'steve': 90,
'john': 12}
>>> d.pop('steve')
90
>>> d
{'brad': 56, 'mary': 34, 'john': 12}

d.clear()
```

Python: Dictionaries

Searching

```
>>> 'mary' in d
True
```

Iterating

```
for key in d.key():
   pass

for val in d.values():
   pass

for (key, val) in d.items()
   pass
```

Python: Functions

```
def function(param1, param2=value, …):
     body
```

```
return
```

```
lambda p1, p2, …: expression(p1, p2, …)
```

```
def grep(string, inputFile, ignoreCase=False):
grep("hero", "novel.txt")
grep("hero", "novel.txt", False)
grep("hero", "novel.txt", True)
grep(inputFile="novel.txt", string="hero")
grep(inputFile="novel.txt", string="hero", True)
grep(string="hero", inputFile="novel.txt", ignoreCase=True)
```

Python: Functions

```
>>> l=[1, 2, 3]
>>> map(lambda x: x*2, l)
[2, 4, 6]
s
>>> l2=[0.1, 0.1, 0.1]
>>> map(lambda x,y: x+y, l, l2)
[1.1, 2.1, 3.1]
```

```
def mysum(l):
    """ This function computes the sum of elements in the given list """
    theSum = reduce(lambda x,y: x+y, l)
    return theSum
```

Python: Files

```
f = open(“file.txt”, “rw”)
f.close()

lines = f.readlines()
f.writelines(lines)

f.write(string)
string = f.read(n)
```

Python: Modules

```
import module_name
module_name.function()

import module_name as new_name
new_name.function()

from module_name import *
from module_name import function
function()

import numpy
numpy.median(l)
from numpy import median
median(l)
```

```
>>> sys.path
['', '/usr/lib/python2.7', '/usr/lib/python2.7/plat-x86_64-linux-gnu', '/usr/lib/python2.7/lib-dynload', '/usr/local/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages', '/usr/lib/python2.7/dist-packages/gtk-2.0', '/usr/lib/pymodules/python2.7']
```

`Python Standard Library:`

http://docs.python.org/2/py-modindex.html

Python: Modules

mystat.py:

```
#!/usr/bin/env python
""" My statistics module """

PI = 3.1415926
def mysum(l):
    theSum = reduce(lambda x,y: x+y, l)
    return theSum

if __name__ == "__main__":
   s = mysum([1,2,3])
   print("test: sum: %f" % (s))
```

proc.py:

```
#!/usr/bin/env python

import mystat

print(mystat.mysum([10,20,30]))
print(mystat.PI)
```

Python: Classes/Objects

A way to encapsulate data and functions.

```
class ClassName(ParentClass):
     classVar = value
     def __init__(self, init_params):
          pass

     def classFn(self, params):
          pass

obj = ClassName(init_params)
print(obj.classVar)
res = obj.classFn(params)
```

```
class Person:
    def __init__(self, name, ident):
        self.name = name
        self.ident = ident

class Course:
    def __init__(self, title, info):
        self.title = title
        self.info = info

class Student(Person):
    def __init__(self, name, id, lCourses):
        Person .__init__(self, name, id)
        self.lCourses = lCourses
        self.dCourse2Info = {}
        self.dCourse2Grade = {}
        self.homework = None

    def learn(self, aCourse):
        self.dCourse2Info[aCourse] = aCourse .
info

    def setGrade(self, aCourse, grade):
        self.dCourse2Grade[aCourse] = grade
```

```
class Teacher(Person):
    def __init__(self, name, id, aCourse):
        Person.__init__(self, name, id)
        self.course = aCourse

    def teach(self, lStudents):
        for student in lStudents:
            student.learn(self.course)

    def grade(self, lStudents):
        for student in lStudents:
            grade = listen(student)
            student.setGrade(self.aCourse,
grade)

    def listen(self, aStudent):
        grade = calcGrade(aStudent.
homework)
        return grade
```

```
math = Course("Math", "knowledge about
math ...")
stats = Course("Statistics", "stats know-
how")

john = Student("John", 12, [math, stats])
mary = Student("Mary", 34, [stats,])
allStudents = [john, mary]

stats_prof  = Teacher("Brad", 56, stats)
math_prof = Teacher("Ellen", 78, math)
allTeachers = [stats_prof, math_prof]

allPeople = allStudents + allTeachers

for p in allPeople:
    print(p.name, p.ident)

stats_prof.teach(allStudents)
```

Python: introspection

```
>>> dir(reduce)
['__call__', '__class__', '__cmp__', '__delattr__', '__doc__', '__eq__', '__format__',
'__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__',
'__module__', '__name__', '__ne__', '__new__', '__reduce__', '__reduce_ex__',
'__repr__', '__self__', '__setattr__', '__sizeof__', '__str__', '__subclasshook__']
>>> reduce.__doc__
'reduce(function, sequence[, initial]) -> value\n\nApply a function of two arguments
cumulatively to the items of a sequence,\nfrom left to right, so as to reduce the
sequence to a single value.\nFor example, reduce(lambda x, y: x+y, [1, 2, 3, 4, 5])
calculates\n((((1+2)+3)+4)+5).  If initial is present, it is placed before the items\nof
the sequence in the calculation, and serves as a default when the\nsequence is empty.'
>>>
```

Python: numpy

- numpy is a Python module wrapping C and Fortran code
- numpy: core/basic functionality, scipy (built on top of NumPy): domain specific functionality (eg: statistics, signal processing, etc)
- numpy routines may be 1000 times faster than functionally equivalent Python code
- numpy is faster than R in some areas and slower in others
- documentation: http://docs.scipy.org/doc/, http://docs.scipy.org/doc/numpy/reference/

Python: numpy

```
>>> import numpy
>>> l=[1, 3, 11, 19, 30]
>>> numpy.average(l)
12.800000000000001
>>> numpy.median(l)
11.0
>>> numpy.amin(l)
1
>>> numpy.amax(l)
30
>>> numpy.std(l)
10.703270528207721
>>> numpy.histogram(l)
(array([2, 0, 0, 1, 0, 0, 1, 0, 0, 1]), array([  1. ,   3.9,   6.8,   9.7,  12.6,  15.5,  18.4,  21.3,
24.2, 27.1,  30. ]))
```

Python and R

rpy2 Python package: execute R code from Python

Examples from rpy.sourceforge.net:

```
import rpy2.robjects as robj
>>> pi = robjects.r['pi']
>>> pi[0]
3.14159265358979

>>> letters = robj.r['letters']
>>> rcode = 'paste(%s, collapse="-")'%(letters.
r_repr())
>>> res = robj.r(rcode)
>>> print(res)
"a-b-c-d-e-f-g-h-i-j-k-l-m-n-o-p-q-r-s-t-u-v-w-x-y-z"
```

```
robj.r('''
        f <- function(r, verbose=FALSE) {
            if (verbose) {
                cat("I am calling f().\n")
            }
            2 * pi * r
        }
        f(3)
        ''')
```

Python and R

rPython R package: execute Python code from R

Example from rpython.r-forge.r-project.org:

```
python.call( "len", 1:3 )
a <- 1:4
b <- 5:8
python.exec( "def concat(a,b): return a+b" )
python.call( "concat", a, b)

python.assign( "a",  "hola hola" )
python.method.call( "a", "split", " " )
```