Python
What's going on?

Stéphane Vialette

LIGM, Université Paris-Est Marne-la-Vallée

September 26, 2012

Outline

1. What is this?
2. Using Python as a calculator

Outline

1 What is this?

2 Using Python as a calculator

What is this?

http://en.wikipedia.org/wiki/Python_%28programming_language%29}

Description

- Python is a general-purpose high-level programming language whose design philosophy emphasizes code readability.
- Python aims to combine "remarkable power with very clear syntax", and its standard library is large and comprehensive.
- Its use of indentation for block delimiters is unusual among popular programming languages.
- It features a fully dynamic type system and automatic memory management, similar to that of Scheme, Ruby, Perl, and Tcl.

What is this?

http://en.wikipedia.org/wiki/Python_%28programming_language%29}

Description

- Python supports multiple programming paradigms, primarily but not limited to *object oriented*, *imperative* and, to a lesser extent, *functional programming* styles.
- Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts.
- The reference implementation of Python (CPython) is free and open source software and has a community-based development model, as do all or nearly all of its alternative implementations.

What is it for?

Network development

- **twisted**: Network framework.
- **PYRO**: Python Remote Objects
- **PyLinda**: Distributed Computing Made Easy.
- ...

Web

- **django**: The Web framework for perfectionists with deadlines.
- **plone CMS**: Open Source Content Management.
- **zope**: Open source application server
- **turbogears**: Web development.
- ...

What is it for?

GUI

- **Tk**: A thin object-oriented layer on top of Tcl/Tk.
- **wxPython**: Cross-platform Python GUI toolkit
- **PyQt**: Python bindings for Trolltech's Qt application framework.
  PyGTK: Python bindings for GTK+ application framework
- ...

GUI

- **PyXML**: XML package for Python.
- **4suite**: Open-source platform for XML and RDF processing.
- **pyRXP**: The fastest XML parser
- ...

What is it for?

Multimedia

- **PyGame**: Python modules designed for writing games (SDL).
- **pyMedia**: Python module for wav, mp3, ogg, avi, divx, dvd, ...
- **Shtoom**: VoIP softphone
- **PiTiVi**: Open source video editor
- ...

PDF

- **ReportLab Toolkit**: Industry-strength PDF generating solution
- **pyPdf**: Pure-Python library built as a PDF toolkit
- ...

What is it for?

and other areas

- **BioPython**: Tools for computational molecular biology.
- **networkx**: Creation, manipulation, and study of the structure, dynamics, and functions of complex networks.
- **NLTK**: The Natural Language Toolkit.
- **PLY**: Implementation of lex and yacc parsing.
- **SciPy**: Open-source software for mathematics, science, and engineering.
- **Simpy**: Simulation in Python.
- **jython**: Implementation of Python written in 100% pure Java.
- **unittest**: Unit testing framework
- ...

What do I need?

Editors

- **emacs**: The extensible, customizable, self-documenting, real-time display editor. `python-mode` is the associated major mode.
- **SPE**: Stani's Python Editor.
- **PyDev**: Python development enviroment for Eclipse.
- **geany**: A fast and lightweight IDE.
- **Boa Constructor**: Cross platform Python IDE.
- **PyPE**: Editor written in Python with the wxPython GUI toolkit.
- **eric**: A full featured Python (and Ruby) editor and IDE, written in python.
- ...

Python interpreter

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56) [GCC 4.4.3] on linux2
>>> print 'hello word'
hello word
>>> import sys
>>> dir(sys)
['__displayhook__', '__doc__', '__egginsert', '__excepthook__', '__name__',
 '__package__', '__plen', '__stderr__', '__stdin__', '__stdout__',
 '_clear_type_cache', '_current_frames', '_getframe', 'api_version', 'argv',
 'builtin_module_names', 'byteorder', 'call_tracing', 'callstats', 'copyright',
 'displayhook', 'dont_write_bytecode', 'exc_clear', 'exc_info', 'exc_type',
 'excepthook', 'exec_prefix', 'executable', 'exit', 'flags', 'float_info',
 'getcheckinterval', 'getdefaultencoding', 'getdlopenflags',
 'getfilesystemencoding', 'getprofile', 'getrecursionlimit', 'getrefcount',
 'getsizeof', 'gettrace', 'hexversion', 'maxint', 'maxsize', 'maxunicode',
 'meta_path', 'modules', 'path', 'path_hooks', 'path_importer_cache', 'platform',
 'prefix', 'ps1', 'ps2', 'py3kwarning', 'pydebug', 'setcheckinterval',
 'setdlopenflags', 'setprofile', 'setrecursionlimit', 'settrace', 'stderr',
 'stdin', 'stdout', 'subversion', 'version', 'version_info', 'warnoptions']
>>>
$
```

Python interpreter

```
$ python --help
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser; also PYTHONDEBUG=x
-E     : ignore PYTHON* environment variables (such as PYTHONPATH)
-h     : print this help message and exit (also --help)
-t     : issue warnings about inconsistent tab usage (-tt: issue errors)
-u     : unbuffered binary stdout and stderr; also PYTHONUNBUFFERED=x
         see man page for details on internal buffering relating to '-u'
         -v     : verbose (trace import statements); also PYTHONVERBOSE=x
                  can be supplied multiple times to increase verbosity
                  -V     : print the Python version number and exit
                           (also --version)
                  -W arg : warning control; arg is
                           action:message:category:module:lineno
                  -x     : skip first line of source, allowing use of
                           non-Unix forms of #!cmd
file   : program read from script file
-      : program read from stdin (default; interactive mode if a tty)
arg ...: arguments passed to program in sys.argv[1:]
```

Hello word.

Python interpreter

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license" for more
information.
>>> s = 'hello word.'
>>> print s
hello word.
>>> import sys
>>> sys.stdout.write(s)
hello word.>>> sys.stdout.write(s + '\n')
hello word.
>>>
$
```

Hello word.

Running a script

```
$ ls helloword.py
helloword.py
$ file helloword.py
helloword.py: ASCII text
$ cat helloword.py
s = 'hello word.'
print s
$ python helloword.py
hello word.
$
```

Hello word.

Standalone script

```
$ ls standalone.py
standalone.py
$ ./standalone.py
bash: ./standalone.py: Permission denied
$ ls -l standalone.py
-rw-r--r-- 1 garulfo garulfo 51 2010-09-27 05:57 standalone.py
$ chmod u+x standalone.py
-rwxr--r-- 1 garulfo garulfo 51 2010-09-27 05:57 standalone.py
$ ./standalone.py
hello word.
$
```

Hello word.

From the Python interpreter

```
$ cat helloword.py
s = 'hello word.'
print s
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>> execfile('helloword.py')
hello word.
>>> import helloword
hello word.
>>>
$
```

Hello word.

Executing Python code

```
$ python
Python 2.6.5 (r265:79063, Apr 16 2010, 13:09:56)
[GCC 4.4.3] on linux2
Type "help", "copyright", "credits" or "license"
for more information.
>>> exec("s = 'hello word.'" + "\n" + "print s")
hello word.
>>> cmd = "s = 'hello word.'" + "\n" + "print s"
>>> print cmd
s = 'hello word.'
print s
>>> exec(cmd)
hello word.
>>>
$
```

Hello word.

From C

```
cat hellowordinC.c#include <Python.h>

int main(int argc, char argv[])
{
    Py_Initialize() ;
    PyRun_SimpleString("s = \"hello  word.\"");
    PyRun_SimpleString("print s");
    return(0);
}
$
$ ls -l /usr/include/python2.6/Python.h
-rw-r--r-- 1 root root 4384 2010-04-16 15:59 /usr/include/python2.6/Python.h
$ ls -l /usr/lib/libpython2.6.so
lrwxrwxrwx 1 root root 17 2010-09-27 06:13 /usr/lib/libpython2.6.so ->
                                           libpython2.6.so.1
$ gcc -o hellowordinC hellowordinC.c -I/usr/include/python2.6 -lpython2.6
$ ./hellowordinC
hello  word.
$
```

Outline

1 What is this?

2 Using Python as a calculator

Numbers

http://docs.python.org/tutorial/introduction.html

Python interpreter

```
>>> 1+2
3
>>> # This is a comment
...
>>> 1+2 # This is a comment
3
>>> (50-5*6)/4
5
>>> # Integer division returns the floor
... 9/2
4
>>> 9/-2
-5
>>>
```

Numbers

Python interpreter

```
>>> pi = 3.14
>>> radius  = 1
>>> # Circumference
... 2 * pi * radius
6.2800000000000002
>>> circumference = 2 * pi * radius
>>> circumference
6.2800000000000002
>>> # Area
... area = pi * radius * radius
>>> area
3.1400000000000001
>>> # Multiple computations
... 2 * pi * radius, pi * radius * radius
(6.2800000000000002, 3.1400000000000001)
>>> # Multiple assigments
... circumference, area = 2 * pi * radius, pi * radius * radius
>>> (circumferencen area)
(6.2800000000000002, 3.1400000000000001)
>>>
```

Assigning several variables simultaneously

```
>>> # Assigning several variables simultaneously
... x = y = z = 0
>>> x
0
>>> y
0
>>> z
0
>>> x, y, z
(0, 0, 0)
>>> # But
... x+2 = y+1 = z = 0
  File "<stdin>", line 1
  SyntaxError: cannot assign to operator
>>> x = y = z = 0
>>> # Comparison
... x == 0
True
>>> # Multiple comparisons
... x==0, y<0, z>=0
(True, False, True)
>>>
```

Variables must be “defined” before they can be used

```
>>> a
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'a' is not defined
>>> a = 1
>>> a
1
>>> # a is an integer
... type(a)
<type 'int'>
>>> if a == 0:
...     b = 1
...
>>> b
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'b' is not defined
>>> if a == 1:
...     b = 1
...
>>> b
1
```

Floating point

Python interpreter

```
>>> # Floating point support
... 1.2 * 3.4 / 4.5
0.9066666666666673
>>> a = 1+2
>>> a, type(a)
(3, <type 'int'>)
>>> # Operators with mixed type operands convert
... # the integer operand to floating point
... a = 1.0+2
>>> a, type(a)
(3.0, <type 'float'>)
>>> (1+2)/3
1
>>> (1.0+2)/3
1.0
>>> (1+2)/3.0
1.0
>>> (1+2+0.0)/3
1.0
>>>
```

Complex numbers

Python interpreter

```
>>> 1j * 1J
(-1+0j)
>>> 1j * complex(0,1)
(-1+0j)
>>> 3+1j*3
(3+3j)
>>> (3+1j)*3
(9+3j)
>>> (1+2j) / (1+1j)
(1.5+0.5j)
>>> (1+2j) * (1+1j)
(-1+3j)
>>> x = 1 + 2j
>>> x
(1+2j)
>>> type(x)
<type 'complex'>
>>>
```

Conversion functions

Python interpreter

```
>>> a = 1
>>> a, type(a)
(1, <type 'int'>)
>>> # Convert to Float
... float(a)
1.0
>>> b = float(a)
>>> b, type(b)
(1.0, <type 'float'>)
>>> # Convert to int
... c = int(b)
>>> c, type(c)
(1, <type 'int'>)
>>> # But
... d = int(b) + 2.3
>>> d
3.2999999999999998
>>> d = int(b + 2.3)
>>> d
3
>>>
```

Conversion functions and complex numbers

Python interpreter

```
>>> a=3.0+4.0j
>>> # Real part?
... int(a)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot convert complex to int
>>> # Imaginary part?
... float(a)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: cannot convert complex to float
>>> # Real part
... a.real, type(a.real), (3.0+4.0j).real
(3.0, <type 'float'>, 3.0)
>>> # Imaginary part
... a.imag, type(a.imag), (3.0+4.0j).imag
(4.0, <type 'float'>, 4.0)
>>> # Magnitude, i.e., sqrt(a.real**2 + a.imag**2)
... abs(a)
5.0
```

“Magic” variable _

Python interpreter

```
>>> 1+2
3
>>> # The last printed expression is assigned to the variable _
... _
3
>>> _+4
7
>>> _
7
>>> _+_, _*_
(14, 49)
>>> _
(14, 49)
>>> a = b = _
>>> a, b, _
((14, 49), (14, 49), (14, 49))
>>> _
((14, 49), (14, 49), (14, 49))
>>>
```

“Magic” variable _

Python interpreter

```
>>> # Don't explicitly assign a value to _, you would create an
... # independent local variable with the same name masking the
... # built-in variable with its magic behavior.
... _ = 1
>>> _
1
>>> 1+2
3
>>> _
1
>>> del _
>>> _
1
>>> 1+2
3
>>> _
3
>>>
```