A mini Python tutorial

v0.03 2009-01-13 - Philippe Lagadec

Here is a mini Python tutorial, for people who want to quickly learn Python basics. It also provides links to more detailed documentation.

For the latest version, see http://www.decalage.info/python/tutorial

Quote: as Philippe Biondi told me a few years ago, "Python is the shortest path between the idea and the running program". Or, as Bruce Eckel said "Life is short, you need Python".

Useful links

1) Download a Python interpreter

- Python official website: http://www.python.org
- Download: http://www.python.org/download

2) Install a Python-aware editor

- on Windows, I recommend PyScripter to get an effective IDE with syntax highlighting, code completion, integrated Python shell and debugger: http://mmm-experts.com/Products.aspx?ProductId=4 or http://code.google.com/p/pyscripter/ for latest versions.
- on other OSes such as Linux or MacOSX: Eclipse + PyDev plugin
- or Spe (very cool but somewhat unstable)
- or simply use the IDLE script provided with the Python interpreter

3) Documentation

- On Windows, the manual provided with the interpreter (Start menu / All Programs / Python / Manual) is usually the most convenient way to access the Python documentation. Use the index tab !
- Online official documentation startpoint: http://www.python.org/doc
- (pretty long) official tutorial: http://docs.python.org/tut/tut.html
- Very quick tutorial: http://www.poromenos.org/tutorials/python?
- Reference manuals: http://docs.python.org/lib/lib.html
- Quickref: http://rgruet.free.fr/#QuickRef
- Useful recipes: http://aspn.activestate.com/ASPN/Cookbook/Python/
- French wiki: http://wikipython.flibuste.net/

Running Python

There are at least two ways to run Python code:

- from the interactive interpreter:
    - **On Unix/Linux/MacOSX**, launch "python" in a shell. Quit with Ctrl+D or type "exit()" when finished.
    - **On Windows**, use Start Menu / Python / Python command line. Alternatively you may run "python" from a CMD window, but since python.exe is not in the PATH environment variable by default you may want to add it, or type its complete path such as "C:\python25\python.exe". Quit with Ctrl+Z or simply close the window when finished.
- or write a script in a ".py" file and launch "python myscript.py"

Variables

Variables are simply names which point to any value or object:

```
a_string = "hello, world"
an_integer = 12
a_float = 3.14
a_boolean = True
```

Print

To print a constant or variable:

```
print "hello, world"
print 12
print (5+3)/2
```

To print several items, use commas (items will be separated by spaces):

```
print "abc", 12, a_float
```

A long statement may be split using backslash:

```
print 'a long statement may be split using backslash', \
     'this is still the same statement', \
     'the end.'
```

Strings

There are 3 syntaxes for string constants:

```
string_single_quotes = 'abc'
string_double_quotes = "abc"
string_triple_quotes = """this is
a multiline
string."""
```

It's useful when we need to include quotes in a string:

```
string1 = 'hello "world"'
string2 = "don't"
```

otherwise we have to use backslashes:

```
string2 = 'don\'t'
```

Other special characters: http://docs.python.org/ref/strings.html

Strings are objects which support many operations:

```
strings = string1 + " : " + string2
strings_uppercase = strings.upper()
```

All string methods: http://docs.python.org/lib/string-methods.html

Slicing (substring extraction):

```
beginning = strings[0:4]
```

More about slicing:

- http://docs.python.org/tut/node5.html#SECTION005120000000000000000
- http://docs.python.org/lib/typesseq.html

Several ways to include an integer into a string:

```
string1 = 'the value is ' + str(an_integer) + '.' # by concatenation
string2 = 'the value is %d.' % an_integer # by "printf-like" formatting
```

With several variables, we need to use parentheses:

```
a = 17
b = 3
string3 = '%d + %d = %d' % (a, b, a+b)
```

To include strings into another string:

```
stringa = '17'
stringb = '3'
stringc = 'a = '+stringa+', b = '+stringb
stringd = 'a = %s, b= %s' % (stringa, stringb)
```

Everything about string formatting: http://docs.python.org/library/stdtypes.html#string-formatting-operations

Convert a string to an integer and vice-versa:

```
i = 12
s = str(i)

s = '17'
i = int(s)
```

Strip spaces at beginning and end of a string:

```
stripped = a_string.strip()
```

Replace a substring inside a string:

```
newstring = a_string.replace('abc', 'def')
```

Important note about strings

A Python string is "immutable". In other words, it is a constant which cannot be changed in place. All string operations create a new string. This is strange for a C developer, but it is necessary for some properties of the language. In most cases this is not a problem.

Comments

It is always useful to add comments to your scripts:

```
# Comments start with "#"
```

Lists:

A list is a dynamic array of any objects.
It is declared with square brackets:

```
a_list = [1, 2, 3, 'abc', 'def']
```

Lists may contain lists:

```
another_list = [a_list, 'abc', a_list, [1, 2, 3]]
```

(in this case, "a_list" is only a pointer)

Access a specific element by index (index starts at zero):

```
elem = a_list[2]
elem2 = another_list[3][1]
```

It's easy to test if an item is in the list:

```
if 'abc' in a_list:
     print 'bingo!'
```

Extracting a part of a list is called slicing:

```
list2 = a_list[2:4] # returns a list with items 2 and 3 (not 4)
```

Other list operations like appending:

```
a_list.append('ghi')
a_list.remove('abc')
```

Other list operations: http://docs.python.org/lib/typesseq.html

Tuples

A tuple is similar to a list but it is a fixed-size, immutable array. This means that once a tuple has been created, its elements may not be changed, removed, appended or inserted.

It is declared using parentheses and comma-separated values:

```
a_tuple = (1, 2, 3, 'abc', 'def')
```

But parentheses are optional:

```
another_tuple = 1, 2, 3, 'abc', 'def'
```

Tip: a tuple containing only one item must be declared using a comma, else it is not considered as a tuple:

```
a_single_item_tuple = ('one value',)
```

A bit more about tuples: http://docs.python.org/library/stdtypes.html#sequence-types-str-unicode-...

Blocks and Indentation (control flow)

Blocks of code are delimited using indentation, either spaces or tabs at the beginning of lines. This is one of the main differences of Python over other languages, and that is the main reason why people love it or hate it. ;-)

Tip: NEVER mix tabs and spaces in a script, it may result in tricky bugs.
From my experience, the safest solution is to **always use 4-spaces indents, never tabs**. (because each editor may convert tabs either to 2, 4 or 8 spaces)

if / elif / else:

```
if a == 3:
     print 'The value of a is:'
     print 'a=3'


if a == 'test':
     print 'The value of a is:'
     print 'a="test"'
     test_mode = True
else:
     print 'a!="test"'
     test_mode = False
     do_something_else()


if a == 1 or a == 2:
     pass # do nothing
elif a == 3 and b > 1:
     pass
elif a==3 and not b>1:
     pass
else:
     pass
```

while loops:

```
 a=1 while a<10:
     print a
     a += 1
```

for loops:

```
for a in range(10):
     print a


my_list = [2, 4, 8, 16, 32]
for a in my_list:
     print a
```

More about control flow keywords: http://docs.python.org/tutorial/controlflow.html

Functions

A function is defined using the "def" keyword:

```
def my_function (arg1, arg2, arg3='default value'):
     print 'arg1 =', arg1
     print 'arg2 =', arg2
     print 'arg3 =', arg3
```

Call it (note that arguments are not strongly typed):

```
my_function (17, 'abc', 3.14)
```

The third argument may be omitted:

```
my_function ('a string', 12)
```

A function may return a value:

```
def fun2():
     print 'fun2'
     return 'any return value'
```

call it:

```
print 'fun2() = %s' % fun2()
```

Script Arguments

It is necessary to import modules from the standard library:

```
import sys
```

Command-line arguments are strings stored in a list sys.argv:

```
n = len(sys.argv)
for i in range(n):
     print 'sys.argv[%d] = "%s"' % sys.argv[i]
```

Files

Open a file for reading:

```
f = open('my_file.txt')
```

open() returns a file object with several methods.
To read a specific number of characters:

```
s = f.read(10)
```

To read the whole file in a string:

```
s = f.read()
```

To close an open file:

```
f.close()
```

To iterate over file lines:

```
f = open('my_file.txt')
for line in f:
     print line
f.close()
```

Open a file for writing:

```
fw = open('out_file.txt', 'w')
fw.write('a line of text\n') # note '\n' is necessary here
fw.write(a_string)
fw.close()
```

Other file operations: http://docs.python.org/lib/bltin-file-objects.html

To act on filenames, for example to remove a file, you need to import standard library modules such as os and os.path:

```
import os, os.path
if os.path.exists('my_file.txt'):
     os.remove('my_file.txt')
```

All file/dir pathnames operations from os and os.path:

- http://docs.python.org/lib/os-file-dir.html
- http://docs.python.org/lib/module-os.path.html

Other useful features

This very short tutorial cannot cover all features of the Python language and its standard library. Everything may be found in the official documentation, either in the Python manual installed on Windows or online. Here is a selection:

- Dictionaries: http://docs.python.org/lib/typesmapping.html or http://docs.python.org/tutorial/datastructures.html#dictionaries
- Modules: http://docs.python.org/tutorial/modules.html
- Classes: http://docs.python.org/tutorial/classes.html
- Regular expressions: http://docs.python.org/lib/module-re.html
- XML parsing: http://www.decalage.info/en/python/etree

Batteries included

Python's standard library is amazingly powerful: http://docs.python.org/library/index.html

Tip on Windows: always keep the Python manual open (Start menu / Python / Python Manuals), and use the index tab to easily find useful modules by typing keywords.

If you cannot find the feature you're looking for, then dive into the Python Package Index, also called PyPI or "Cheeseshop": http://pypi.python.org