Introduction to Python - Part I
CNV Lab

Paolo Besana

22 - 26 January 2007

This quick overview of Python is a reduced and altered version of the online tutorial written by Guido Van Rossum (the creator of Python), that can be found at the URL:
http://docs.python.org/tut/tut.html

1 Interpreter

A great advantage of Python is the option of launching the interpreter and directly executing commands. This can save a lot of time when you are developing software: when you are not sure about something, you can often test it directly from the interpreter, without the overhead of having a small program to launch from the command line. The interpreter is launched by typing the following command in the shell:

```
python
```

Once the interpreter has been launched, lines previously inserted in the interpreter shell can be recalled pressing the ↑ cursor button. To exit the interpreter, press `Control-D`.
To launch a program written in Python, type the following command from the shell:

```
python programname.py
```

Open a terminal window. Create a new directory called "`MyPython`":
`s0xxxxx: mkdir MyPython`
Go to this directory:
`s0xxxxx: cd MyPython`
Launch Python Interpreter:
`s0xxxxx: python`

2 Basic Programming

Unlikely most other programming languages, in Python, the formatting of the code has a meaning: there are no braces or begin/end delimiters around blocks of code: groups of statements are indented under a header. Blocks of code can be nested, using increasing indentation. There are no end-of-line symbols, like the semicolon (;) in Java or C: the new line marks the end of a line.

2.1 Basic Types

Numbers

The interpreter can be used as a simple calculator.
▷ *Enter the following expression in the python interpreter you have just launched:*

```
>>> 2+2
>>> 3*2
```

Strings

Strings can be enclosed in single or double quotes.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> 'hello'
>>> 'how\'s going?'
>>> "how's going?"
```

Strings can be enclosed in pairs of triple quotes, either """ or '''. In this case it is not necessary to escape end of lines.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> print """
... This is a string that when
... printed mantains its format.
... """
```

Strings can be concatenated using the + operator, and can be repeated with the operator *.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> 'Hello ' + ', ' + 'how are you'

>>> 'help' + '!'*5
```

The individual characters of a string can be accessed using indices. The first character has index 0. Substrings can be specified with slice notation: two indices separated by colon. When using slices, indices can be thought as pointing between character, instead of pointing to characters: the left edge of the first character is 0 and so on.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> word="hello"
>>> word[0]

>>> word[2]

>>> word[2:4]
```

```
>>> word[:2]

>>> word[2:]
```

Negative indices start to count from the right.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> word[-1]

>>> word[-2:]
```

`upper()` and `lower()` convert the characters of a string into upper case and lower case:

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> 'Hello!'.upper()

>>> 'Hello!'.lower()
```

`strip()` function removes the spaces at the beginning and at the end of the string:

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> '  Hello !   '.strip()
```

Strings cannot be modified once they are created. Trying to modify a substring results in an error. However, it is easy to create a new string concatenating substrings from other strings.

Lists

There are different types of compound structures in Python. The most versatile is the *list*.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> L = ['mon','tue','wed','thu','fri']
>>> L
```

List items can also be accessed using indices. They can be sliced and concatenated like strings.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> L[0]

>>> L[3]

>>> L[1:3]

>>> L + ['sat','sun']
```

It is possible to check the membership of an element in a list using the keyword `in`.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> 'wed' in L
>>> 'sun' in L
```

Items can be added at the end of a list using the method `append(item)` of the list object.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> L3 = []
>>> L3.append(1)
>>> L3.append(2)
>>> L3
```

It is possible to measure the length of a list using the built-in function `len(list)`.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> len(L)
```

2.2 Control Structures

Python offers the usual control flow statements, as other languages like Java or C. The `for` loop is more powerful than in most of the other languages.

if

▷ *Type the lines introduced by "> > >" and by "..."*
[When asked, insert a binary sequence (such as 00101 or 10010)]

```
>>> x=rawinput("Enter a binary sequence : ")
>>> if (x[0]=='0'):
...    print "Starts with 0"
... elif (x[0] == '1'):
...    print "Starts with 1"
... else:
...    print "Error"
```

It is possible to create more complex conditions using the keywords `and` for conjunction and `or` for disjunction.

while

The `while` statement repeats the indented block of code as long as the condition is true:

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> x = 10
>>> while (x > 0):
...     print x
...     x = x - 1
```

for

The `for` statement iterates over the items of any sequence (such as strings or lists).

▷ *Type the lines introduced by "> > >" and by "..."*
[the variable `L` has already been assigned by you (see page 3)]

```
>>> for x in L:
...     print x
...
```

To iterate over a sequence of numbers the built-in function `range()` is used to automatically generate them.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> range(5,10)
>>> for x in range(3):
... print x
...
>>> for x in xrange(3):
... print x
...
```

(Here `xrange()` is a variant of `range()` that is optimized for use in `for` loops; it generates each element as it is needed rather than requiring a big chunk of memory at the start.)

EXERCISE 1

Type:

```
>>> Lst = ['how', 'why', 'however', 'where', 'never']
```

Write a loop in the python interpreter (which should still be open) that prints for every element in the list:

- ▷ a star symbol `*`
- ▷ the first two letters from the element
- ▷ the whole element

producing something like:

```
* ho how
* wh why
...
```

EXERCISE 2

Modify the previous loop to print a star ('*') in front of the elements that start with "wh" and a plain space ' ' in front of the others, producing something like:

```
  ho how
* wh why
...
```

2.3 Functions

A function is introduced by the keyword `def`. It must be followed by the function name and by the list of parameters in parenthesis. The statements that form the function body start the next line, and are indented. The first line can be an optional string that describes the function. This string can be used by automatic generators of documentation.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> def square(value):
...     """Return the square of the value"""
...     return value*value ...
>>> square(4)
```

EXERCISE 3

Write a function `checkPrefix(list, prefix)` in the python interpreter (which should still be open). The function wraps the loop created in the previous exercise: when called it must print the content of the list, adding a star in front of the elements that start with the prefix.

2.4 List Comprehensions

List comprehensions provide a concise and clear way to create lists, which is often quite important for writing readable and maintainable Python code. Each list comprehension consists of an expression followed by a for clause, then zero or more `for` or `if` clauses. The result will be a list resulting from evaluating the expression in the context of the `for`and`if`clauses that follow it. If the expression would evaluate to a tuple, it must be parenthesized.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> freshfruit = [' banana', ' loganberry ', 'passion fruit ']
>>> [w.strip() for w in freshfruit]
```

```
>>> v = [2, 4, 6]
>>> [3*x for x in v]
```

The `if` clause is used to filter the elements that have to appear in the generated list:

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> [3*x for x in v if x > 3]

>>> [3*x for x in v if x < 2]

>>> [[x,x**2] for x in v]

>>> [(x, x**2) for x in v]
```

It possible to combine items from different lists, obtaining a Cartesian product of elements, using a `for` clause for each list :

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> v1 = [2, 4, 6]
>>> v2 = [4, 3, -9]
>>> [x*y for x in v1 for y in v2]

>>> [x+y for x in v1 for y in v2]

>>> [v1[i]*v2[i] for i in range(len(v1))]

>>> [str(round(355/113.0, i)) for i in range(1,6)]
```

EXERCISE 4

Using list comprehension:

▷ Write code that converts the elements in the following list:

```
L4 = ['hello', 'how', 'are', 'you']
```

into upper case

▷ Write code that generates a list containing only the even numbers, divided by 2, from the following list:

```
L5 = [3, 52, 21, 43, 12, 18, 17]
```

HINT: a number is even if the remainder of a division by 2 is equal to 0 (the remainder is obtained using %: `4%2==0`, `5%2==1`)

▷ Write code that generates, from the following list, a list containing pairs composed of the values and their parity values:

```
L5 = [3, 52, 21, 43, 12, 18, 17]
```

Each pair is a list composed of two elements: the value itself, and true if value is even or false otherwise

2.5 Modules

Programs can become long, and it is a good approach to divide them into more manageable units. In Python, programs can be divided into *modules*. The modules can be imported directly into the interpreter or into other programs. Python has a very large library of predefined functions and classes that can be imported and used.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> import math
```

When we call the function, it is necessary to prefix it with the module name, to avoid name conflicts:

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> math.pi
>>> math.pow(2,5)
```

We can also import only the function we need, and in this case it is not necessary to prefix it with the module name.

▷ *Type the lines introduced by "> > >" and by "..."*

```
>>> from math import pow
>>> pow(2,5)
```

If you modify a file, you need to reload the module using `reload(modulename)`, although this doesn't always do what you might expect because itdoes not reload anything the module depends on.

▷ *End your interpreter session (CTRL+D)*