Programming for NLP : the Python language

Guy Perrier

1 - First steps with Python

1. Presentation of Python
2. Expressions and types
3. Instructions, variables and assignment
4. Python scripts

1.1 – Presentation of Python

- Python was created at the beginning of 1990 years by **Guido van Rossum.**
- Python is a programming language distributed under **free license** and usable on all current platforms (Windows, Linux, Mac).
- The version used in this course is 2.7 and a new version 3 is available but it is not compatible with the previous versions.
- For more information about Python, see the official web site : http://www.python.org/

1.1 – Presentation of Python

- Python is an **interpreted language** by opposition to **compiled languages**.
- A program in an interpreted language is read step by step by a special program related to the language and the computing platform : the **interpreter**. The interpreter runs the program as it reads it step by step.
- A program in a compiled language is translated into a language of a lower level by a special program related to the language and the computing platform : the **compiler**. The original program is called the **source program** and the resulting program the **object program**. The object program is then executed.
- Python can run in **interactive mode** or in **automatic mode**. In interactive mode, every line is executed as soon as it is entered on the keyboard. In automatic mode, a program, called a script, is stored in a file (with the extension .py) and then it is executed at once.
- **Idle** is a graphical interface which facilitates the use of Python.

1.2 - Expressions and types

- An **expression** is an object that can be evaluated : one can compute its **value**. An expression also has a **type**.

- An expression can be **atomic**. Its value generally identifies with it.

```
>>> type(2)
<type 'int'>

>>>type(10000000000000000000)
<type 'long'>

>>> type(2.34)
<type 'float'>

>>> type(2.4E5)
<type 'float'>
```

```
>>> type('bonjour')
<type 'str'>

>>> type("bonjour")
<type 'str'>

>>> type(False)
<type 'bool'>

>>> type(True)
<type 'bool'>
```

1.2 - Expressions and types

- A **compound expression** is composed from other expressions and operators acting on them. Its value is computed from the values of its components and from the meaning of the operators.

```
>>> "the" + "table"
'the table'

>>> 4.5*4
18.0

>>> True and False
False

>>> 2>3
False

>>> range(2)
[0,1]
```

Call of function

1.2 - Expressions et types

- When an expression includes several operators, **priority rules** give the order in which operations are performed. **Parentheses** allow this order to be modified.

```
>>> 2.5+3/4
2.5

>>> (2.5+3)/4
1.375

>>> False or True and False or True
True

>>> 'a' == 'b' or False
False

>>> 3 in range(3)
False
```

1.2 - Expressions and types

Operator	Description
lambda	Lambda expression
or	Boolean OR
and	Boolean AND
not	Boolean NOT
in, not in	Test of membership
is, is not	Test of identity
<, <=, >, >=, <>, !=, ==	Comparisons
|	OR bit to bit
^	XOR bit to bit
&	AND bit to bit
<<, >>	Bit shifting
+,-	Addition and substraction
*, /, %	Multiplication, division, remainder
+, -	Positive, opposite
~	NOT bit to bit
**	Power
id. attribute	Attribute reference
id [*index*]	Access to an array entry
id [*index : index*]	Access to an array slice
id (arguments)	Function call

1.2 - Expressions and types

- For a given operator, the types of its operands must be among the possible types expected by the operator.

```
>>> "table"- "ta"
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    "table"- "ta"
TypeError: unsupported operand type(s) for -: 'str' and 'str'

>>> "table" + 2
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    "table" + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

- A non standard use of an operator is possible with arithmetic operators after **type conversion** of the operands.

```
>>> True + 12
13
>>> type(True+12)
<type 'int'>

>>> 12 + 3.0/4
12.75
>>> type(12 + 3.0/4)
<type 'float'>
```

1.2 - Expressions and types

- Boolean operators have a specific behavior: they can be used with any types of operands. The following operand values are interpreted as false: False, None, numeric zero of all types, and empty strings and containers. All other values are interpreted as true.
  - ✓ The operator “not” yields True if its argument is false, False otherwise.
  - ✓ The expression “x and y” first evaluates *x*; if *x* is false, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.
  - ✓ The expression “x or y” first evaluates *x*; if *x* is true, its value is returned; otherwise, *y* is evaluated and the resulting value is returned.

```
>>> “a” and True
True
>>> “a” or True
a
>>> True or “a”
True
>>> 5 and  2
2
>>> 0 and  2
0
```

1.3 - Instructions, variables and assignment

- A **program** is a combination of **instructions**. Expression evaluations are instructions.
- Value storing is performed with an **assignment instruction**, which has the following form :

  variable = expression

- A **variable** is a memory location named with an **identifier** and used to store a **value**.
- The variable takes the type of the used expression until a new assignment (**dynamic typing**).
- Variables can be used in the building of expressions.
- By using a variable in both sides of an assignment, the value of a variable can be modified using its old value. For this, the following abbreviation is available :

  variable op = expression which means : *variable = variable op expression*

1.3 - Instructions, variables and assignment

```
>>> x = 2==3
>>> type(x)
<type 'bool'>
>>> x
False

>>> x=3*x
>>> x
0
>>> type(x)
<type 'int'>

>>> x=  "the"
>>> type(x)
<type 'str'>
>>> x
‘the’

>>> x=3*x
>>> x
’thethethe’

>>> x+= ‘ table’
>>> x
‘thethethe table’
```

1.4 - Python scripts

- In automatic mode, a Python program, called a script, must be stored in a file (with the .py extension). The advantage with respect to the interactive mode is that it allows the program to be kept in memory to be modified and re-used.

- A script is executed either trough a graphical interface like Idle, or in command line with command python followed by the identifier of the file containing the script.

- A script can be written with a standard editor. At the beginning of the script, one may indicate the character encoding with a comment. For example, for the Latin-1 encoding, the comment is : *# -*- coding:latin-1 -*-* . For UTF-8, the comment is : *# -*- coding:utf-8 -*-*

- A script is generally composed of a sequence of instructions. For its understanding, it is important to insert **comments**. A comment starts with the # symbol and ends with a new line. The Python interpreter ignores comments.

1.4 - Python scripts

- Example of Python script stored in file *surface.py*

```
# Program for computing the surface of a rectangle : surface = length x width



length = input(Input the length in cm :  ") # input of the length  in cm with the keyboard
width = input("Input  the width in cm :  ") # input of the width in cm with the keyboard
surface = length * width #computation of the surface in cm2
print "Surface of the rectangle : ", surface, " cm2" # display of the surface
```

- A possible execution of *aire.py*.

```
>>>
Input a length in cm :  8.5
Input a width in cm :  4
Surface of the rectangle :  34.0  cm2
>>>
```

1.5 Exercises

1. Determine if every expression is well-formed. If yes, determine its type and its value.

   a) *13 % 2 ** 3 * 2 / 3*

   b) *15/4 * 'a' + 'b'*

   c) *2==3 + "good"*

   d) *2 == 5 and not 3 + 2*

   e) *"a" + "b" or 5 and 0 in range(2)*

2. The following instructions are assumed to be executed in order. For each one, give the modification of the variable content that it entails.

```
x  = (5 and 2) % 3
y  = 2.5 + x * 3 % 2
x *= y / 5 ** 2
z =  2 * x <= y
z  *= "10"
x =  3 * '10' > z
```

1.5 Solutions of the exercises

1.

a) *((13 % (2 ** 3))* 2)/ 3 → 3 : int*

b) *(15/4 * 'a')+ 'b' → 'aaab' : Str*

c) *(2==3)+ "good" → incorrect : an integer cannot be added to a string*

d) *(2 == 5) and (not (3 + 2))→ False : bool*

e) *(("a" + "b") or (5 and (0 in range(2))) → 'ab' : Str*

2.

instructions	x	y	z
x = (5 and 2) % 3	2		
y = 2.5 + x * 3 % 2		2.5	
x *= y / 5 ** 2	0.0102040816327		
z = 2 * x <= y			True
z * = "10"			"10"
x = 3 * "10" > z	True		