W1-S4-C4-print-en-python3

December 15, 2014

1 `print` entre python2 et python3

1.1 Complément - niveau avancé

Ce complément est destiné aux étudiants qui s'intéressent à python3, et qui voudraient dès le départ prendre l'habitude d'utiliser `print` comme elle existe en python3, c'est-à-dire comme une fonction.

N'hésitez pas à passer ce complément si vous n'êtes pas concerné par ce sujet.

Notez toutefois que les différences entre python2 et python3 ne se limitent pas à `print`. Si vous commencez python aujourd'hui, il peut en effet être une bonne idée d'utiliser `print` comme une fonction, mais restez conscient que ce cours est sur python2, et quíl ne suffit pas de mettre des parenthèses autour de `print` pour transformer votre code en python3.

1.1.1 Les comportements de `print`

Après ce préambule, sachez que la différence la plus visible entre python2 et python3 est certainement avec `print`:

python2 En python2 `print` est une **instruction**

```
>>> import sys
>>> sys.version
'2.7.8 (default, Jul 13 2014, 17:11:32) \n[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)]'
>>> print "Hello", "World"
Hello World
```

et comme c'est une instruction on ne peut pas comparer son résultat avec autre chose - ici 0

```
>>> 0 == print "Hello", "World"
  File "<stdin>", line 1
    0 == print "Hello", "World"
              ^
SyntaxError: invalid syntax
```

python3 Alors qu'en python3 `print` est une **fonction** (et donc une **expression**)

```
>>> import sys
>>> sys.version
'3.4.1 (default, Sep 20 2014, 19:44:17) \n[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)]'
>>> print("Hello","World")
Hello World
```

et comme toute expression elle retourne une valeur qu'on peut comparer - ici encore avec 0

```
>>> 0 == print("Hello","World")
Hello World
False
```

1.1.2 Compatibilité avec `__future__`

Il existe en python2 une couche de compatibilité, le module `__future__`, dont le but est de gommer les différences entre les deux versions. Ainsi si vous le souhaitez vous pouvez écrire du code python2 qui "voit" `print` comme une fonction en important ceci

```
from __future__ import print_function
```

Après quoi vous pouver utilisez `print` comme si vous écriviez du python3

```
>>> import sys
>>> sys.version
'2.7.8 (default, Jul 13 2014, 17:11:32) \n[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)]'
>>>
>>> from __future__ import print_function
>>>
>>> print("Hello","World")
Hello World
```

1.1.3 Remarque

Si vous utilisez python2 et que **vous ne mentionnez pas cet import**, vous pouvez avoir l'**impression** qu'il est inutile:

```
>>> import sys
>>> sys.version
'2.7.8 (default, Jul 13 2014, 17:11:32) \n[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)]'
>>>
>>> print("Hello")
Hello
```

En fait ce qui se passe ici, c'est que les parenthèses sont considérées dans leur rôle habituel de groupement dans les expressions; cela rejoint la discussion que l'on aura sur les tuples; en fait les parenthèses ne jouent aucun rôle ici, exactement comme dans

```
In []: ("Hello")
```

Si bien qu'avec plusieurs arguments - plus exactement avec l'illusion d'appeler `print` avec plusieurs arguments:

```
In []: print("Hello", "World")
```

les parenthèses présentes dans la sortie vous indiquent qu'en fait on a contruit un objet qui est un tuple, et qu'on l'a passé à l'**instruction** `print`; bref, on ne peut pas utiliser `print` dans une expression

```
In []: 0 == print("Hello", "World")
```

Une fois l'import chargé, tout fonctionne vraiment comme en python3

```
In []: from __future__ import print_function

       0 == print("Hello", "World")
```