W1-S3-C2-notebooks-et-interpreteur

December 15, 2014

1 Modes d'exécution

Nous avons donc à notre disposition plusieurs façons d'exécuter un programme python. Nous allons les étudier plus en détail.

Quoi
Avec quel outil
ligne à ligne
avec IDLE ou python interactif
fichier complet
`python <fichier>.py`
par fragments
dans un notebook

Pour cela nous allons voir le comportement d'un tout petit programme python lorsqu'on l'exécute sous ces trois environnements.

L'objectif de cet exercice est de vous convaincre que - heureusement - les trois environnements se comportent de la même façon, avec toutefois une petite différence quant au niveau de détail de ce qui se trouve imprimé.

Essentiellement, lorsqu'on utilise l'interpréteur en mode interactif - ou sous IDLE - à chaque fois que l'on tape une ligne, le résultat est **calculé** (on dit aussi **évalué**) **et imprimé**

Par contre lorsqu'on écrit tout un programme, on ne peut plus imprimer le résultat de toutes les lignes, cela produirait un flot d'impression beaucoup trop important.

Enfin en ce qui concerne le notebook, le comportement est un peu hybride entre les deux, en ce sens que seul le **dernier résultat** de la cellule est imprimé.

1.0.1 L'interpréteur python interactif

Le programme choisi est très simple, c'est le suivant

```
10 * 10
20 * 20
30 * 30
```

Voici comment se comporte l'interpréteur interactif quand on lui soumet ces instructions

```
~ $ python
Python 2.7.7 (default, Jun  2 2014, 01:41:14)
[GCC 4.2.1 Compatible Apple LLVM 5.1 (clang-503.0.40)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> 10 * 10
100
>>> 20 * 20
400
>>> 30 * 30
900
```

```
>>> exit()
~ $
```

Noter que pour terminer la session, il nous faut "sortir" de l'interpréteur en tapant `exit()`
On peut aussi taper `Control-D` sous linux ou MacOS.
Comme on le voit ici, l'interpréteur imprime **le résultat de chaque ligne**. On voit bien apparaître toutes les valeurs calculées, 100, 400, puis enfin 900.

1.0.2 Sous forme de programme constitué

Voyons à présent ce que donne cette même séquence de calculs dans un programme complet. Pour cela il nous faut tout d'abord fabriquer un fichier, que la tradition veut avec un suffixe en `.py`, en utilisant par exemple un éditeur de fichier. Le résultat doit ressembler à ceci:

```
~ $ cat foo.py
10 * 10
20 * 20
30 * 30
~ $
```

Exécutons à présent ce programme:

```
~ $ python foo.py
~ $
```

On constate donc que ce programme ne fait rien. En tous cas selon toute apparence.
Ce qui se passe en réalité, c'est que les 3 valeurs 100, 400 et 900 sont bien calculées, mais comme aucune instruction `print` n'est présente, rien n'est imprimé et le programme se termine sans signe apparent d'avoir réellement fonctionné.
Ce comportement peut paraître un peu déroutant au début, mais comme nous l'avons mentionné c'est tout à fait délibéré. Un programme fonctionnel faisant facilement plusieurs milliers de lignes, voire beaucoup plus, il ne serait pas du tout réaliste que chaque ligne fasse l'objet d'une impression automatique comme c'est le cas en mode interactif.

1.0.3 Dans un notebook

Voici à présent le même programme dans un notebook

```
In []: 10 * 10
       20 * 20
       30 * 30
```

Lorsqu'on exécute cette cellule (rappel : sélectionner la cellule, et utiliser le bouton en forme de flêche vers la droite, ou entrer **"Shift+Enter"** au clavier), on obtient une seule valeur dans la rubrique 'Out', 900, qui correspond **au résultat de la dernière ligne.**

1.0.4 Utiliser `print`

Ainsi pour afficher un résultat intermédiaire, on utilise l'instruction `print`. Nous verrons cette instruction en détail dans les semaines qui viennent, mais en guise d'introduction disons seulement qu'on la fait suivre d'une liste d'expressions comme ceci:

```
In []: a = 10
       b = 20

       print a, b
```

On peut naturellement mélanger des objets de plusieurs types, et donc mélanger des strings et des nombres pour obtenir un résultat un peu plus lisible; en effet lorsque le programme devient gros, il est important de savoir à quoi correspond une ligne dans le flot de toutes les impressions. Aussi on préfèrera quelque chose comme:

```
In []: print "a=", a, "et b=", b
```

Une pratique courante consiste d'ailleurs à utiliser les commentaires pour laisser dans le code les instructions `print` qui correspondent à du debug (c'est-à-dire qui ont pu être utiles lors de la mise au point et qu'on veut pouvoir réactiver rapidement).

1.0.5 Utiliser `print` pour "sous-titrer" une affectation

Remarquons enfin et surtout le fait que l'affectation à une variable ne retourne aucun résultat.

C'est à dire en pratique que si on écrit

```
In []: a = 100
```

même une fois l'expression évaluée par l'interpréteur, aucune ligne `Out[]` n'est ajoutée.

C'est pourquoi il nous arrivera parfois, et notamment lorsque l'expression est complexe, pour rendre plus explicite la valeur qui vient d'être affectée, d'écrire alors plutôt

```
In []: a = 100 ; print a
```

Notez bien que cette techinque est uniquement pédagogique et n'a absolument aucun autre intérêt dans la pratique, il n'est **pas recommandé** de l'utiliser en dehors de ce contexte.