Partie 3
Progresser

Chapitre 3-1
Trucs et astuces

1. Introduction

Dans les deux premières parties de cet ouvrage, vous avez découvert Python, puis vous avez appris à copier et à reproduire du code Python.

Cette troisième et dernière partie est beaucoup moins structurée que les autres. Il s'agit plus de montrer certains points et certaines possibilités du langage de manière croissante.

Au début, nous verrons quelques trucs et astuces et quelques notions déjà évoquées, mais présentées de manière plus approfondie. Vous trouverez aussi quelques exemples, dont certains sont volontairement récréatifs et didactiques, alors que d'autres ont une application bien réelle et pourraient vous rendre service.

Les deux derniers chapitres traitent d'outils et de logiciels écrits en Python.

Le chapitre Aller plus loin avec Python parle de quelques outils Python utilisables avec un peu d'investissement.

Tandis que le dernier chapitre évoque des logiciels écrits en Python, mais d'une dimension supérieure, où il est nécessaire de s'investir un peu plus pour bénéficier de leurs apports. Cependant, soyez sûr que le jeu en vaut la chandelle, et certains développeurs ont réellement apporté quelque chose, au moins pendant un temps, à l'informatique en général. Il serait dommage de ne pas profiter de ce travail exceptionnel.

Et même s'il ne s'agit que de découvrir de quoi ces produits sont capables, en le faisant, vous aurez du respect pour ces développeurs et leur œuvre, ce qui est la moindre des choses.

2. Adapter le copier/coller d'un tableur pour un wiki

Il s'agit d'un petit truc, qui peut rendre service et qui permet d'appréhender certaines choses.

Lorsque l'on souhaite copier/coller une sélection de cellules d'un tableur vers un wiki, on se retrouve avec une tabulation comme séparateur et il faut donc reformater le tableau en mode wiki, ce qui peut être fastidieux.

Mais grâce à Python, il est possible de modifier le contenu du clipboard (presse-papier).

Le clipboard, sous Linux, n'est pas quelque chose d'évident et, par exemple, pour le faire fonctionner, il a fallu installer un outil s'appelant xclip (`sudo apt install xclip`).

Il faut aussi installer le module `clipboard` avec '`pip install clipboard`'.

Comment l'utilise-t-on ?

Il faut tout simplement sélectionner ses données dans la feuille de calcul et lancer un petit script.

Ensuite, il faut effectuer un copier/coller dans le wiki… c'est magique.

Copier depuis le tableur -> Lancer le script Python -> Coller dans le wiki.

Voici un exemple du résultat :

```
===> Sans le script python
Colonne 1    Colonne 2
100    200
101    201

===> Avec le script python
^Colonne 1    ^Colonne 2    ^
|100    |200    |
|101    |201    |
|       |
```

Le script ne fait que rajouter des chapeaux '^' en en-tête de colonne et des pipes '|' comme séparateur de ligne, mais quand il y a beaucoup de lignes, c'est pratique.

Voici le script :

```
# fichier : copie_colle.py

##
## Modifie le copier coller
## pour passer d'un tableau scalc a un wiki
## Idée originale de JC P.

import clipboard

## on récupère les données du clipboard
data = clipboard.paste()

## on les traite
tmp = data.split('\n')

entete = tmp.pop(0).split('\t')

tmp_data = [ '^'+x for x in entete ]+['^']

new_data = '\t'.join(tmp_data)+'\n'

for l in tmp:
```

```
    d = l.split('\t')
    tmp_data = [ '|'+x for x in d ]+['|']
    new_data += '\t'.join(tmp_data)+'\n'

# on remet dans le clipboard
clipboard.copy(new_data)
```

3. Déballage avec Python (unpacking)

Cela tient plus du sucre syntaxique que du vrai "truc et astuce", mais cela peut être très pratique.

Attention, c'est le genre de syntaxe qui ne saute pas forcément aux yeux et donc il y a un risque de perte de visibilité. Donc il faut bien commenter ce genre de syntaxe.

Dans les premiers chapitres où l'on découvre l'opérateur d'affectation '=', on découvre aussi la possibilité d'affecter plusieurs variables :

```
>>> a = b = c = 1
>>> print(a,b,c)
1 1 1

>>> a,b,c = 1,2,3
>>> print(a,b,c)
1 2 3
```

Mais il y a encore mieux avec l'opérateur '*'.

Il est aussi possible d'affecter une variable et/ou une liste.

Voici quelques exemples :

```
>>> liste = list(range(0,9))
>>> liste
[0, 1, 2, 3, 4, 5, 6, 7, 8]

>>> *x, y, z = liste

>>> print("x=",x," y=",y, " z=",z)
x= [0, 1, 2, 3, 4, 5, 6]  y= 7  z= 8

>>> x, *y, z = liste
```

```
>>> print("x=",x," y=",y, " z=",z)
x= 0  y= [1, 2, 3, 4, 5, 6, 7]  z= 8

>>> x,y,*z = liste

>>> print("x=",x," y=",y, " z=",z)
x= 0  y= 1  z= [2, 3, 4, 5, 6, 7, 8]
```

Et cela fonctionne aussi avec les boucles, si par exemple vous avez une liste de listes, mais dont les éléments n'ont pas tous le même nombre d'éléments…

Voici un exemple :

```
# fichier : unpack1.py

liste = [
   list(range(0,10)),
   list(range(0,6)),
   list(range(0,12)),
   ]

print(liste)

for x,*y,z in liste:
   print( "x=", x)
   print( "y=", y)
   print( "z=", z)
```

Ce qui donne :

```
[[0, 1, 2, 3, 4, 5, 6, 7, 8, 9], [0, 1, 2, 3, 4, 5], [0, 1, 2, 3,
4, 5, 6, 7, 8, 9, 10, 11]]

x= 0
y= [1, 2, 3, 4, 5, 6, 7, 8]
z= 9

x= 0
y= [1, 2, 3, 4]
z= 5

x= 0
y= [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
z= 11
```

Pour rappel, il est possible d'ignorer une partie d'une affectation avec l'utilisation de '_'

```
x, y, z, _ = liste ## on ne veut que les 3 premiers éléments
```

ou même pire, ne vouloir que la dernière ligne d'un fichier :

```
>>> *_, last = open("/etc/fstab")

>>> print(last)

UUID=e01ec561-2a07-4872-b469-15b189307f55 none     swap    sw     0    0
```

4. L'underscore et Python

L'underscore est un caractère multi-utilisation dans le langage Python.

Voici quelques trucs que l'on peut rencontrer dans du code.

4.1 Dans l'interpréteur

Underscore conserve la dernière valeur dans l'interpréteur et on peut l'utiliser comme une variable.

```
>>> [ x for x in range(0,5)]
[0, 1, 2, 3, 4]
>>> _
4
>>> _ * 2
8
```

4.2 Pour ignorer des valeurs

Quand, dans une liste, on ne veut que certains éléments.

```
>>> premier, _ , dernier   = (1,2,3)
>>> print(premier, dernier)
1 3
```

Ou mieux avec l'opérateur '*'.

```
>>> premier,*_, dernier = list(range(0,10))
>>> print(premier, dernier)
0 9
>>> _
[1, 2, 3, 4, 5, 6, 7, 8]
```

4.3 Dans les boucles

Il est possible de l'utiliser comme indice de boucle :

```
>>> for _ in range(5):
...   print(_)
...
0
1
2
3
4
```

4.4 Pour la séparation des milliers

Pour améliorer la lisibilité sur les grands nombres.

50_000_000 est équivalent à 50 millions.

```
>>> 1024*1024*1024*1024
1099511627776

>>> 1_099_511_627_776
1099511627776

>>> 1_099_511_627_776 / 1024 / 1024 / 1024
1024.0
```