Formatage de chaînes de caractères

Complément - niveau basique

On désigne par formatage les outils qui permettent d'obtenir une présentation fine des résultats, que ce soit pour améliorer la lisibilité lorsqu'on s'adresse à des humains, ou pour respecter la syntaxe d'un outil auquel on veut passer les données pour un traitement ultérieur.

L'instruction `print`

Nous avons jusqu'à maintenant presque toujours utilisé l'instruction `print` pour afficher nos résultats. Comme on l'a vu, celle-ci réalise un formatage sommaire, en ceci qu'elle insère un espace entre les valeurs qui lui sont passées.

La seule subtilité notable concernant `print` est que, si on termine l'instruction par une virgule, on évite le saut de ligne qui est autrement ajouté automatiquement

```
print "une", "seule", "ligne"
print "une", "autre",
print "ligne"
```

Il faut remarquer aussi que `print` "sait" imprimer n'importe quel type d'objet; nous l'avons déjà fait avec les listes et les tuples, et nous verrons plus tard comment définir la présentation à utiliser lorsqu'on définit ses propres types.

La méthode `format`

On rencontre assez vite les limites de `print`. D'une part, il peut être nécessaire de formater une chaîne de caractères sans nécessairement vouloir l'imprimer, ou en tous cas pas immédiatement. D'autre part les espaces ajoutés peuvent être plus néfastes qu'utiles.

On peut dans ces cas-là recourir à la méthode `format`, qui est définie sur les objets `str`, et qui s'utilise comme suit. Mais définissons d'abord quelques données à afficher.

```
age = 35
nom = "Lambert"
prenom = "Jacques"
```

Nous avons choisi des objets simples, mais exactement comme avec `print`, `format` peut accepter n'importe quel type d'objet.

On peut alors préparer une mise en forme de ces données comme ceci

```
"{}, {}, {}ans".format(nom, prenom, age)
```

Dans cet exemple le plus simple, les données fournies sont affichées en lieu et place des {}, dans l'ordre où elles sont fournies.

Cela convient bien lorsqu'on a peu de données; si par la suite on veut changer l'ordre par exemple des nom et prénom, on peut bien sûr échanger l'ordre des arguments passés à format, ou encore utiliser la liaison par position, comme ceci

```
"{1}, {0}, {2}ans".format(nom, prenom, age)
```

Dans la pratique toutefois, cette forme est assez peu utile, on lui préfère souvent la liaison par nom qui se présente comme ceci

```
"{le_prenom}, {le_nom}, {l_age}ans".format(le_nom=nom, le_prenom=prenom, l_age=age
)
```

Dans ce premier exemple de liaison par nom, nous avons délibérément utilisé des noms différents pour les données externes et pour les noms apparaissant dans le format, pour bien illustrer comment la liaison est résolue, mais on peut tout aussi bien faire tout simplement

```
"{prenom}, {nom}, {age}ans".format(nom=nom, prenom=prenom, age=age)
```

Il est possible de mélanger la liaison par position et la liaison par nom comme ceci

```
"{1}, {0}, {age}ans".format(nom, prenom, age=age)
```

avec la réserve, toutefois que les noms doivent apparaître après les arguments liés par position; les raisons pour ceci vous apparaîtront clairement dans quelque temps; aussi la forme suivante n'est pas acceptée

```
"{1}, {0}, {age}ans".format(age=age, nom, prenom)
```

```
File "<ipython-input-46-9a8e8e541bf2>", line 1
    "{1}, {0}, {age}ans".format(age=age, nom, prenom)
SyntaxError: non-keyword arg after keyword arg
```

Enfin, il arrive qu'on ait besoin de spécifier plus finement la façon dont une valeur doit être affichée; c'est typiquement le cas avec les valeurs flottantes, pour lesquelles la précision avec laquelle on les montre vient au détriment de la lisibilité.

```
from math import pi
entier = 100
```

Voici quelques formes équivalentes pour obtenir une valeur de pi arrondie

```
"avec seulement 3 chiffres apres la virgule {:.3f} - entier {}".format(pi, entier)
```

```
"avec seulement 3 chiffres apres la virgule {flottant:.3f} - entier {entier:d}".for
mat(flottant=pi, entier=entier)
```

Nous vous invitons à vous reporter à la documentation de `format` (https://docs.python.org/2.7/library/string.html#formatstrings) pour plus de détails sur les formats disponibles et aux nombreux exemples disponibles dans la documentation (https://docs.python.org/2.7/library/string.html#format-examples) .

Voici qui conclut notre courte introduction à la méthode `format`, dont on rappelle toutefois pour être bien clair qu'elle **ne réalise pas d'impression**, il faut donc la coupler à `print` si l'impression est souhaitée.

La version précédente du formatage : l'opérateur `%`

`format` a été en fait introduite relativement tardivement dans python, pour remplacer la technique que nous allons présenter maintenant.

Il est bien entendu conseillé d'utiliser exclusivement `format` pour le code que vous écrirez, mais étant donné le volume de code qui a été écrit avec l'opérateur `%`, il nous a semblé important d'introduire rapidement cette construction ici.

Le principe de l'opérateur % est le suivant. On élabore comme ci-dessus un "format" c'est-à-dire le patron de ce qui doit être rendu, auquel on passe des arguments pour "remplir" les trous. Voyons les exemples de tout à l'heure rendus avec l'opérateur `%`

```
"%s, %s, %sans" % (nom, prenom, age)
```

Les personnes ayant été exposées à C et C++ verront d'où venait cette conception des formats. Reconnaissons toutefois que le code en question n'est pas très lisible, dans l'exemple ci-dessus on peut avoir du mal à analyser le "sans" qui apparaît comme un mot à première vue.

Quoi qu'il en soit, on pouvait également avec cet opérateur recourir à un mécanisme de liaison par nommage, en passant par un dictionnaire. Pour anticiper un tout petit peu sur cette notion que nous verrons très bientôt, voici comment:

```
variables =  {'le_nom':nom, 'le_prenom': prenom, 'l_age':age}
"%(le_nom)s, %(le_prenom)s, %(l_age)sans" % variables
```