W2-S8-C2-outils-sur-chaines

December 14, 2014

1 Les outils de base sur les strings

1.1 Complément - niveau intermédiaire

1.1.1 Lire la documentation

Même après des années de pratique, il est difficile de se souvenir de toutes les méthodes travaillant sur les chaînes de caractères. Aussi il est toujours utile de recourir à la documentation embarquée:

```
In []: help(str)
```

Nous allons tenter ici de citer les méthodes les plus utilisées; nous n'avons le temps que de les utiliser de manière très simple, mais bien souvent il est possible de passer en paramètre des options permettant de ne travailler que sur une sous-chaîne, ou sur la première ou dernière occurrence d'une sous-chaîne; nous vous renvoyons à la documentation pour obtenir toutes les précisions utiles.

1.1.2 Découpage - assemblage : `split` et `join`

On l'a vu dans la vidéo, la paire `split` et `join` permet de découper une chaine selon un séparateur pour obtenir une liste, et à l'inverse de reconstruire une chaîne à partir d'une liste.

`split` permet donc de découper:

```
In []: 'abc=:=def=:=ghi=:=jkl'.split('=:=')
```

Et à l'inverse

```
In []: "=:=".join(['abc', 'def', 'ghi', 'jkl'])
```

Attention toutefois si le séparateur est un terminateur, la liste résultat contient alors une dernière chaîne vide.

```
In []: 'abc;def;ghi;jkl;'.split(';')
```

Qui s'inverse correctement cependant

```
In []: ";".join(['abc', 'def', 'ghi', 'jkl', ''])
```

1.1.3 Remplacements : `replace`

`replace` est très pratique pour remplacer une sous-chaîne par une autre, avec une limite éventuelle sur le nombre de remplacements

```
In []: "abcdefabcdefabcdef".replace("abc", "zoo")
```

```
In []: "abcdefabcdefabcdef".replace("abc", "zoo", 2)
```

Plusieurs appels à `replace` peuvent être chaînés comme ceci

```
In []: "les [x] qui disent [y]".replace("[x]", "chevaliers").replace("[y]", "Ni")
```

1.1.4 Nettoyage : `strip`

On pourrait par exemple utiliser `replace` pour enlever les espaces dans une chaîne, ce qui peut être utile pour "nettoyer" comme ceci

```
In []: " abc:def:ghi ".replace(" ", "")
```

Toutefois bien souvent on préfère utiliser `strip` qui ne s'occupe que du début et de la fin de la chaîne, et gère aussi les tabulations et autres retour à la ligne

```
In []: " une chaine avec des truc qui depassent \n".strip()
```

1.1.5 Rechercher une sous-chaîne

Plusieurs outils permettent de chercher une sous-chaîne. Le plus simple est `find` qui renvoie le plus petit index où on trouve la sous-chaîne

```
In []: "abcdefcdefghefghijk".find("def")
```

```
In []: "abcdefcdefghefghijk".find("zoo")
```

`rfind` fonctionne comme `find` mais en partant de la fin de la chaîne

```
In []: "abcdefcdefghefghijk".rfind("fg")
```

La méthode `index` se comporte comme `find` mais en cas d'absence elle lève une exception (nous verrons ce concept plus tard) plutôt que de renvoyer -1

```
In []: "abcdefcdefghefghijk".index("def")
```

La méthode `count` compte le nombre d'occurrences d'une sous-chaîne

```
In []: "abcdefcdefghefghijk".count("ef")
```

Signalons enfin les méthodes de commodité suivantes

```
In []: "abcdefcdefghefghijk".startswith("abcd")
```

```
In []: "abcdefcdefghefghijk".endswith("ghijk")
```

S'agissant des deux dernières, remarquons que
chaine.startswith (sous_chaine) $\iff$ chaine.find(sous_chaine) == 0
chaine.endswith (sous_chaine) $\iff$ chaine.rfind(sous_chaine) == (len(chaine)-len(sous_chaine))

1.1.6 Capitalisation

Voici pour conclure quelques méthodes utiles qui parlent d'elles-mêmes

```
In []: "monty PYTHON".upper()
```

```
In []: "monty PYTHON".lower()
```

```
In []: "monty PYTHON".swapcase()
```

```
In []: "monty PYTHON".capitalize()
```

```
In []: "monty PYTHON".title()
```

1.1.7 Pour en savoir plus

Tous ces outils sont documentés en détail ici (en anglais)

1.1.8 Expressions régulières

Signalons enfin que python propose également un module d'expressions régulières, qui permettent de faire des calculs beaucoup plus élaborés. Cette technique bien connue a été rendue populaire historiquement au travers notamment du langage Perl.

Cette technique est très puissante, et permet d'aborder des cas qui vont bien au-delà de ce qui est présenté ici, pour reconnaître les chaînes qui obéissent à des règles lexicales précises. Étant d'une utilisation très spécifique, nous avons choisi d'aborder ce sujet en dernière semaine lors de la session consacrée aux sujets avancés.