Python avancé

GCC ! – Prologin

2019

Durant ce TP, vous allez voir des notions un peu plus avancées de Python, avec des outils tels que les `sets` et les `dict`.

Les ensembles - `set`

Les `sets` sont des structures un peu similaires à une liste : ils contiennent un *ensemble* d'éléments. À la différence des listes, chaque élément est *unique*, et il n'y a pas de notion d'ordre. On s'intéresse uniquement à l'appartenance d'un élément au set. Un set contenant 1, 2 et 3 est identique au set contenant 1, 3 et 2, et il ne peut pas contenir 1 plusieurs fois.

L'intérêt d'utiliser un set par rapport à une liste est généralement en termes de performance. Ajouter, supprimmer et surtout vérifier l'appartenance d'un élément dans un set sont bien plus efficaces que sur une liste. On peut aussi éviter de devoir gérer et supprimer les doublons quand nécessaire.

Création d'un set

Il existe plusieurs méthodes pour créer un set. La plus simple est de le créer directement à partir des éléments, de manière similaire à la création d'une liste en écrivant les éléments. Ici, on utilise des accolades `{}` à la place des crochets `[]`.

Une autre méthode pour créer un set est l'utilisation de la fonction `set`, qui peut prendre en paramètre une liste. Cela permet de créer un set contenant l'ensemble des éléments de cette liste.

Les exemples suivants montrent différentes manières de créer des sets, qui sont tous identiques et équivalents au set contenant 1, 2 et 3.

```
1 {1, 2, 3}
2 {1, 2, 2, 3}
3 {3, 2, 1}
4 set([1, 2, 2, 3])
```

Manipuler un set

Le premier traitement que l'on peut vouloir effectuer sur un set est la vérification d'appartenance. Celle-ci se fait en écrivant `x in s`, qui vaut `True` si l'élément `x` est dans le set `s`, ou `False` sinon.

Pour ajouter ou supprimer un élément `x` à un set `s`, on peut écrire respectivement `s.add(x)` ou `s.remove(x)`.

On peut aussi vouloir parcourir chaque élément du set dans une boucle. Cela se fait de la même manière que pour une liste : `for x in s`. La boucle s'exécute avec `x` qui prend chaque valeur présente dans le set `s`.

Exemple

Cet exemple vient de l'interpréteur Python. Une ligne commençant par `>>>` représente du code Python, et si ce code a une valeur, il sera affiché en dessous.

```
>>> s = {1, 2, 3}
>>> s.add(42)
>>> s
{1, 2, 3, 42}
>>> s.add(42)
>>> s
{1, 2, 3, 42}
>>> 10 in s
False
>>> 2 in s
True
```

Quelques exercices

Pour vous entraîner, vous pouvez écrire quelques fonctions en manipulant les sets :

- Une fonction qui affiche les 5 premières lignes *uniques* écrites par l'utilisateur (lues avec `input()`), en continuant de lire en cas de doublon
- Lister les permutations possibles d'une chaîne de caractères, sans afficher plusieurs fois la même permutation

Les dictionnaires - `dict`

Un `dict` est une structure qui permet de représenter des associations entre des *clés* et des *valeurs*. Un dictionnaire contient un ensemble de clés, et une valeur est associée à chaque clé.

Création d'un dictionnaire

Un dict peut être créé de différentes manières. Pour en créer un directement à partir des valeurs, on peut l'écrire ainsi :

```
{
    "one": 1,
    "two": 2
}
```

Cela créé un set associant la clé `"one"` à la valeur `1`, et `"two"` à `2`.

Manipulation d'un dictionnaire

Le principal outil pour manipuler un dictionnaire est l'utilisation des crochets `dictionnaire[clé]`. Cela permet d'accéder à la valeur associée à la clé si elle existe, et peut aussi être utilisé pour écrire une nouvelle valeur.

Attention, essayer de lire la valeur d'une clé qui n'est pas présente dans le dictionnaire provoque une *exception*[1]. Pour régler ce problème, écrire `dictionnaire.get(clé, défaut)` permet d'accéder à la valeur associée à la clé si elle existe, et renvoie le défaut si elle n'existe pas.

Pour supprimer un élément associé à une clé `x` d'un dictionnaire `dictionnaire`, on peut écrire `del dictionnaire[x]`.

On peut également manipuler un dictionnaire de manière similaire à un set, avec `x in dictionnaire` ou `for x in dictionnaire`. Dans ce cas, le dictionnaire se comporte comme un set de ses *clés*.

Exemple

```
>>> d = {"one" : 1}
>>> d
{'one': 1}
>>> d["two"] = 2
>>> d["two"]
2
>>> d["one"] = "something else"
>>> d
{'one': 'something else', 'two': 2}
>>> d.get("not here", "default")
'default'
```

1. Similaire à un crash du programme pour l'instant, demandez à un organisateur si vous voulez plus de détails

```
>>> del d["one"]
>>> d
{'two': 2}
```

Exercices

Écrivez la fonction `histogramme(texte)` qui renvoie l'histogramme du texte entré sous forme de dictionnaire. Les caractères présents dans l'entrée sont les clés, et le nombre de fois où le caractère est présent est la valeur associée. Par exemple :

```
>>> histogramme("bonjour")
{'b': 1, 'o': 2, 'n': 1, 'j': 1, 'u': 1, 'r': 1}
```

En bonus pour cet exercice, vous pouvez également vous renseigner sur l'utilisation d'un `defaultdict`, présent dans `collections`.