Chapitre 12. Dictionnaires (hors-programme)

12.1 Introduction

Nous avons étudié les listes et les chaînes de caractères, qui permettent d'accéder à leurs éléments à l'aide d'indices :

```
>>> L = [34,56,43,4]
>>> print(L[2])
43
>>> s = "abcd"
>>> print(s[1])
"b"
```

Les indices apportent ainsi une numérotation des éléments, qui sont donc dans un certain ordre, numérotés à partir de zéro.

Les dictionnaires (`dict`) vont plus loin que cela : leurs éléments peuvent être numérotés de manière complètement arbitraire. `{}` est le dictionnaire vide, et l'on peut alors y ajouter des éléments de manière semblable à l'accès dans une liste, sauf que l'on peut utiliser n'importe quel indice :

```
>>> d = {}
>>> d[23] = 1
>>> d[42] = 12
>>> d[-12] = 123
>>> print(d[-12])
123
```

Si l'on regarde le contenu du dictionnaire :

```
>>> d
{23: 1, 42: 12, -12: 123}
```

On constate que l'on retrouve dedans à la fois les éléments et leur indice (que l'on appelera plutôt clé). On peut aussi directement écrire le dictionnaire de la même façon :

```
>>> d = {
  23: 1,
  42: 12,
  -12: 123
}
>>> print(d[-12])
123
```

Les indices n'ont en fait pas besoin d'être entiers :

```
>>> d[1.2] = 1234
```

Ni même des nombres en fait !

```
>>> d["abc"] = 12345
```

Ils sont ainsi très pratiques pour exprimer des relations.

On peut parcourir un dictionnaire avec une boucle `for`, c'est par contre la clé que l'on obtient, pour accéder à l'élément lui-même il faut utiliser le dictionnaire :

```
>>> for x in d:
      print(x, d[x])
23 1
42 12
-12 123
1.2 1234
abc 12345
```

12.2 Exercices

Exercice 12.2.1 On souhaite écrire une fonction `contientDoublon(L:list) -> bool` qui teste si la liste `L` contient deux fois la même valeur.

- Écrire une première version qui utilise deux boucles `for`. Quelle est sa complexité ?
- Voici une version qui utilise un dictionnaire :

```
def contientDoublon(L:list) -> bool:
  d = {}
  for x in L:
    d[x] = False
  for x in L:
    if d[x] == True:
      return True
    d[x] = True
```

- Remarquez comme elle ressemble à la fonction `voisinsDistincts` de l'exercice 10.3.6.
- Sachant que l'accès dans un dictionnaire contenant n éléments coûte $log(n)$, quelle est la complexité de `contientDoublon` ?

Exercice 12.2.2 On souhaite écrire une fonction `apparitions(text:str) -> dict` calculant le nombre d'apparition des différentes lettres dans un texte. L'idée est de partir du dictionnaire vide, puis pour chaque lettre du texte mettre la valeur 0 dans le dictionnaire en utilisant la lettre comme indice, puis pour chaque lettre du texte ajouter 1 à la valeur dans le dictionnaire en utilisant la lettre comme indice.

Quelle est la complexité de `apparitions` ?

12.3 L'essentiel du chapitre

Le dictionnaire vide est `{}`

On peut ajouter à un dictionnaire des éléments avec n'importe quelle valeur d'indice :

```
>>> d = {}
>>> d[23] = 1
>>> d[42] = 12
>>> d[-1] = 123
>>> d[1.2] = 1234
>>> d["abc"] = 12345
```

On peut le parcourir avec une boucle `for`, c'est par contre la clé que l'on obtient :

```
>>> for x in d:
      print(x, d[x])
23 1
42 12
-12 123
1.2 1234
abc 12345
```