Chapitre (ALGO) 5 Dictionnaires

Le test de programmes peut être une façon très efficace de montrer la présence de bugs, mais il est désespérément inadéquat pour prouver leur absence

— **Edsger DIJKSTRA**

Résumé & Plan

Les listes et chaînes étaient des structures ordonnées : on accédait aux éléments à l'aide d'entiers (éventuellement négatifs). Seulement parfois on aura besoin d'associer des éléments quelconques d'un ensemble K à d'autres éléments d'un ensemble V. Une solution est de numéroter les éléments de K, puis d'utiliser une liste, mais cette numérotation complexifie inutilement le problème. La structure de dictionnaire va nous permettre d'éviter cela.

- Les chapitres d'Informatique sont composés de cours et d'exercices intégrés. Le cours sera projeté au tableau.
- Il n'est pas attendu que toute la classe aborde tous les exercices. Traitez donc en priorité les exercices présents dans la liste donnée à chaque début de séance.
- Exercices 💣 / **Pour aller plus loin** : exercices plus difficiles, ou plus techniques. À ne regarder que si les autres sont bien compris.

Fichier externe ?

NON *pas de fichier externe dans ce TP*

1 GÉNÉRALITÉS

Les dictionnaires sont des collections **non ordonnées** d'objets. On remplace les valeurs entières d'indexage, par des clés (donc des objets plus généraux).

1.1 Définition

- Un *dictionnaire* est une donnée composite qui n'est **pas ordonnée**. De manière plus formelle, un dictionnaire n'est ni plus ni moins d'une application d'un ensemble K (les clefs) vers un ensemble V (les valeurs).
- Il fonctionne par un système de **clé:valeur**, c'est-à-dire associe à chaque clef une valeur. L'analogue des clefs pour les listes est finalement les entiers de $\mathbb{Z}$.
- En python, les clés, comme les valeurs, peuvent être de types différents.

Dans la pratique, on définit un dictionnaire à l'aide d'accolades.

Définir un dictionnaire en Python

```
>>> dressing = {"pantalons":3,"pulls":4,"tee-shirts":8}
>>> dressing["pulls"] # le nombre de pulls
4
>>> vocabulaire = {"navigateur":"browser", "précédent":"back", "suivant":"forward"} # un dictionnaire de traduction
>>> vocabulaire["suivant"]
'forward'
>>> AlanTuring = {"naissance":(23,6,1912), "décès":(12,6,1954), "lieu naissance":"Londres", "lieu décès":"Wilmslow"}
>>> AlanTuring["décès"]
(12, 6, 1954)
>>> len(dressing) # le nombre de clefs
3
```

Exercice 1 | [Solution] On considère le dictionnaire `AlanTuring` suivant :

```
AlanTuring = {"naissance":(23,6,1912), "décès":(12,6,1954), "lieu naissance":"Londres", "lieu décès":"Wilmslow"}
```

Écrire une instruction permettant de répondre aux questions suivantes.

1. Quelle est l'année de naissance d'Alan TURING ?

2. Combien d'années Alan TURING a-t-il vécu ?

3. Est-il mort et né dans la même ville ?

Attention aux clefs

Une liste ne peut servir de clef de dictionnaire : de manière général seuls les objets non mutables — comme les tuple par exemple, qui existent d'ailleurs uniquement pour cette raison — peuvent servir de clef d'un dictionnaire.

```
>>> D = {[1, 2]:"a"} # NON
Traceback (most recent call last):
  File "<input>", line 1, in <module>
TypeError: unhashable type: 'list'
>>> D = {(1, 2):"a"} # OUI
```

1.2 Méthodes keys(), values() et items()

- La méthode `keys()` permet de parcourir les clés.
- La méthode `values()` permet de parcourir les valeurs.
- La méthode `items()` permet de parcourir les couples clé-valeur.

(Parcourir un dictionnaire) Par défaut, si on utilise une syntaxe du type `for k in D` où `D` est un dictionnaire, la variable `k` va parcourir les clefs. On peut aussi choisir de parcourir les valeurs, mais ce besoin est plus rare.

```
for k in D:
    ... # k est une clef
```

```
for k in D.keys():
    ... # k est encore une clef
```

```
for v in D.values():
    ... # v est une valeur
```

```
for k, v in D.items():
    ... # k : clef, v : valeur
```

Exemple 1

```
>>> for k in dressing:
...     print("Il y a", dressing[k], k)
...
Il y a 3 pantalons
Il y a 4 pulls
Il y a 8 tee-shirts
>>> for k in dressing.keys():
...     print("Il y a", dressing[k], k) # fait la même chose
...
Il y a 3 pantalons
Il y a 4 pulls
Il y a 8 tee-shirts
>>> for v in dressing.values():
...     print(v)
...
3
4
8
>>> for k, v in dressing.items():
...     print ("il y a ", v, k)
...
il y a  3 pantalons
il y a  4 pulls
il y a  8 tee-shirts
```

1.3 Création et modification

(Création) En python, on crée un dictionnaire vide par l'instruction :

```
>>> D = {} # ou encore :
>>> D = dict()
```

On peut aussi taper directement les éléments du dictionnaire :

```
>>> dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
```

(Tester l'existence d'une clef) Pour tester si `clef` est déjà présente dans le dictionnaire `D`. On utilise :

```
if k in D:
    ...
```

Par exemple,

```
>>> "pantalons" in dressing
True
>>> "tutu" in dressing
False
```

(Ajouter/modifier/supprimer une association) De manière générale, on utilise l'instruction

`D[clef] = valeur`

deux cas se présentent alors :

- si `clef` est déjà une clef du dictionnaire, alors l'ancienne valeur est **remplacée** par `valeur`,
- si `clef` n'est pas déjà une clef du dictionnaire, alors le couple `clef:valeur` est ajoutée au dictionnaire.

On peut supprimer une association avec l'instruction `del D[clef]`. On peut aussi modifier une clef existante *via* des opérations classiques sur les variables (par exemple, `D[clef] += 1` si la valeur en question est un entier, `D[clef].append(...)` si c'est une liste, *etc.*).

```
>>> dressing = {"pantalons":3, "pulls":4, "tee-shirts":8}
>>> dressing["chaussettes"] = 12
>>> dressing["pantalons"] = 5
>>> dressing
{'pantalons': 5, 'pulls': 4, 'tee-shirts': 8, 'chaussettes': 12}
>>> del dressing["pantalons"]
>>> dressing
{'pulls': 4, 'tee-shirts': 8, 'chaussettes': 12}
```

Exercice 2 | [Solution] On considère le dressing suivant *(à taper dans la console ou l'éditeur)* :

```
dressing = {"pantalons":5, "pulls":4, "tee-shirts":8}
```

1. Écrire (dans la console), trois instructions permettant : d'augmenter de 2 le nombre de pantalons, de diminuer de 1 le nombre de pulls et de remettre à zéro le nombre de tee-shirts. Réafficher le dictionnaire pour vérification.
2. Écrire une fonction d'en-tête `achat(habit, dressing)`, qui :
   - si `habit` est déjà présent dans `dressing`, augmente son nombre de 1.
   - Sinon elle ajoute le couple clef/valeur `habit : 1` au dictionnaire `dressing`.

2 ALGORITHMES CLASSIQUES SUR LES DICTIONNAIRES

2.1 Dictionnaire vers liste de couples

Exercice 3 | Liste d'associations [Solution] Écrire une fonction d'en-tête `couples(D)` prenant en argument un dictionnaire `D`, et renvoyant une liste de tuples correspondant aux couples clef/valeur. Par exemple, si `D = {"pantalons":5, "pulls":4, "tee-shirts":8}`, la fonction devra renvoyer la liste :

```
[("pantalons",5), ("pulls",4), ("tee-shirts",8)]
```

2.2 Mémoïsation

Exercice 4 | Sans doublon amélioré [Solution] Dans cet exercice, on souhaite revenir sur une fonction classique concernant les listes (calculer une version sans doublon d'une liste), mais en l'améliorant. Rappelons que dans cette fonction, on :

- crée une nouvelle liste `L_sd` (comme *« liste sans doublon »*) initialisée à la liste vide.
- On parcourt la liste initiale `L`, et on ajoute l'élément en question dans `L_sd` s'il n'y était pas déjà.
- Lors de cette dernière étape, l'instruction `in` qui teste l'appartenance à `L_sd` peut être coûteuse ; en effet, on parcours à chaque fois toute la liste `L_sd` pour savoir si un élément est déjà présent !

Pour pallier à ce problème, on peut créer un dictionnaire `D` qui au début sera initialisé ainsi :

```
D = {x:False for x in L}
```

et à chaque ajout dans `L_sd`, on met à jour le dictionnaire en passant la valeur associée à `True` : ceci permet d'économiser des parcours de `L_sd`. En effet, au moment de savoir si un élément a déjà été ajouté à `L_sd`, plutôt que de parcourir la liste, on regarde simplement la valeur de l'élément dans le dictionnaire.

Écrire une fonction `sans_doublon(L)` programme ce principe. Par exemple, `sans_doublon([2, 3, 3, 1, 2])` renverra `[2, 3, 1]`.

2.3 Occurences et indices

Exercice 5 | Dictionnaire des occurences [Solution]

On souhaite écrire une fonction d'en-tête `dico_occur(L)` qui pour une liste `L` donnée renvoie un dictionnaire ayant pour clés les différents éléments de la liste et pour valeur associée le nombre de fois où cet élément est présent dans `L`. Par exemple, détaillons sur un exemple le principe.

- Considérons `L = [1,5,1,2,5,5,3,2]`. L'élément `1` apparaît `2` fois, l'élément `5` apparaît 3 fois, ..., l'appel `dico_occur(L)` doit donc renvoyer le dictionnaire `{1:2,5:3,2:2,3:1}`. *Notez que sur cet exemple les clés sont donc des entiers et pas des chaînes de caractères.*
- Par concevoir cette fonction, on propose la démarche suivante :
  - ◇ on initialise un dictionnaire vide `D`,
  - ◇ puis on parcourt chaque élément `x` de la liste `L` : si `x` n'est pas déjà présent comme clé dans le dictionnaire `D`, on on ajoute cette clé au dictionnaire en lui associant la valeur `1`; et si au contraire `x` apparaissait déjà en tant que clé dans le dictionnaire on modifie la valeur associée en lui ajoutant `1` (pour tenir compte de cette nouvelle occurrence de la valeur).

1. Écrire la fonction `dico_occurences`. *On pourra compléter la fonction ci-après.*

```
def dico_occur(L):
    D = _____
    for x in L:
        if x not in D:
            D[x] = _____
        else :
            _____
    return D
```

La fonction fonctionne-t-elle pour les différents caractères composants une chaine de caractères et leur nombre d'occurences ?

2. **[Application : les anagrammes]**
   - Deux dictionnaires sont considérés comme *égaux* s'ils contiennent les mêmes éléments, avec les mêmes valeurs pour les mêmes clés (on rappelle qu'il n'y a pas d'ordre dans un dictionnaire). On peut tester directement dans Python l'égalité de deux dictionnaires à l'aide de ==. Par exemple :

```
>>> D_1 = {"a" : 2, (2) : 3}
>>> D_2 = {(2) : 3, "a" : 2}
>>> D_1 == D_2
True
>>> D_1["a"] -= 1
>>> D_1 == D_2
False
```

   - Une chaîne `c1` est un *anagramme* d'une chaîne `c2` s'il existe une permutation des lettres de `c1` donnant `c2`. Par exemple `sauce` est une anagramme de `cause`.

À l'aide de la question précédente, et des deux points précédents, créer alors une fonction `anagramme(c1, c2)` qui teste si la chaîne `c1` est une anagramme de `c2`.

Exercice 6 | Dictionnaire des indices [Solution]

1. Écrire une fonction d'en-tête `dico_indices(L)` qui pour une liste `L` donnée renvoie un dictionnaire ayant pour clés les différents éléments de la liste et pour valeur associée la liste des indices où l'élément est présent dans `L`.
   Par exemple, pour la liste `L = [1, 5, 1, 2, 5, 5, 3, 2]` où l'élément `1` apparaît en indices `0`, `2` et l'élément 5 apparaît en indices `1`,`4`,`5` ..., l'appel `dico_indices(L)` doit renvoyer le dictionnaire `{1:[0, 2],5:[1, 4, 5],2:[3, 7],3:[6]}`.
2. Comment obtenir le dictionnaire des occurences à partir de celui des indices ?

Exercice 7 | Élément le plus fréquent [Solution]

1. Écrire une fonction d'en-tête `plusfrequent(L)`, utilisant la fonction de l'exercice précédent, prenant en argument une liste `L`, et renvoyant l'élément apparaissant le plus fréquemment dans une liste, ainsi que sa fréquence d'apparition. Cette fonction fonctionne-t-elle pour une chaîne ? *On pourra compléter la fonction ci-après.*

```
def plusfrequent(L):
    D = dico_occur(L)
    k_maxi = _____________
    v_maxi = _____________
    for k in D:
        v = D[k]
        if _____________ > _____________:
            k_maxi = _____________
            v_maxi = _____________
    return k_maxi, _____________
```

2. **[Application à un système de votes]** Une classe de BCPST1 d'un certain lycée (ladite classe souhaite garder son anonymat) décide d'élire son professeur de sciences préféré. Les noms des professeurs sont stockés dans une liste de candidats (appelée `candidats`) qui contient des chaînes de caractères. La fonction `rd.choice` permet de choisir au hasard un élément d'une liste (selon une loi uniforme).

■■ Codage des noms et choix

```
>>> import random as rd
>>> candidats = ["CAhyerre", "JHarter", "NLescure"]
>>> rd.choice(candidats) # choix au hasard
'NLescure'
>>> rd.choice(candidats) # choix au hasard
'CAhyerre'
```

2.1) Créer une liste `urne`, qui est une liste de 80 choix de vote (correspondant aux choix des élèves sur les deux classes), en utilisant la fonction `rd.choice` présentée ci-dessus.

2.2) Conclure.

Exercice 8 | 💣 Construction d'un index pour un texte

[Solution] Soit `c` une chaîne de caractères contenant des mots séparés par un espace (typiquement une phrase), créer une fonction d'en-tête `index(c)` qui renvoie un dictionnaires de clefs les mots de `s`, et de valeurs la liste des indices où ce mot apparaît. *Indication : On pourra utiliser la méthode `split` sur les chaînes comme présenté ci-desous*

```
>>> c = "Les BCPST1 vous êtes toujours là ?"
>>> liste_mots = c.split()
>>> liste_mots
['Les', 'BCPST1', 'vous', 'êtes', 'toujours', 'là', '?']
```

Tester par exemple sur la chaîne :

```
c = "le temps est beau le ciel est bleu le lycée est beau et bleu
"
```

3 SOLUTIONS DES EXERCICES

Solution (exercice 1) [Énoncé]

```
>>> AlanTuring["naissance"][2]
1912
>>> AlanTuring["décès"][2]-AlanTuring["naissance"][2]
42
>>> AlanTuring["lieu naissance"] == AlanTuring["lieu décès"]
False
```

Solution (exercice 2) [Énoncé]

1.
```
>>> dressing = {"pantalons":5, "pulls":4, "tee-shirts":8}
>>> dressing["pantalons"] += 2
>>> dressing["pulls"] -= 1
>>> dressing["tee-shirts"] = 0
>>> dressing
{'pantalons': 7, 'pulls': 3, 'tee-shirts': 0}
```

2.
```
def achat(habit, dressing):
    if habit in dressing:
        dressing[habit] += 1
    else:
        dressing[habit] = 1
```

```
>>> achat("tee-shirts", dressing) # la variable dressing est \
↪ modifiée
>>> dressing
{'pantalons': 7, 'pulls': 3, 'tee-shirts': 1}
>>> achat("tutu", dressing)
>>> dressing
{'pantalons': 7, 'pulls': 3, 'tee-shirts': 1, 'tutu': 1}
```

Solution (exercice 3) [Énoncé]

```
def couples(D):
    L = []
    for k in D:
        L.append((k, D[k]))
    return L
```

```
>>> couples({"pantalons":5, "pulls":4, "tee-shirts":8})
[('pantalons', 5), ('pulls', 4), ('tee-shirts', 8)]
```

Solution (exercice 4) [Énoncé]

```
def sans_doublon(L):
    L_sd, D = [], {x:False for x in L}
    for x in L:
        if not D[x]:
            # x n'est pas dans L_sd
            L_sd.append(x)
            D[x] = True
    return L_sd
```

```
>>> sans_doublon([2, 3, 3, 1, 2])
[2, 3, 1]
```

Solution (exercice 5) [Énoncé]

1.
```
def dico_occur(L):
    D = {}
    for x in L:
        if x not in D:
            D[x] = 1
        else :
            D[x] += 1
    return D
```

```
>>> L = [1, 5, 1, 2, 5, 5, 3, 2]
>>> dico_occur(L)
{1: 2, 5: 3, 2: 2, 3: 1}
```

La fonction s'adapte aux chaînes puisque l'on se repère de la même façon dans une chaîne et une liste. Par exemple

```
>>> c = "toto"
>>> dico_occur(c)
{'t': 2, 'o': 2}
```

2.
```
def anagramme(c1, c2):
    D_1 = dico_occur(c1)
    D_2 = dico_occur(c2)
    return D_1 == D_2
```

```
>>> anagramme("cause", "sauce")
True
>>> anagramme("abcde", "sauce")
False
```

Solution (exercice 6) [Énoncé]

1.
```
def dico_indices(L):
    D = {}
    for i in range(len(L)):
        if L[i] not in D:
            D[L[i]] = [i]
        else :
            D[L[i]].append(i)
    return D
```
```
>>> L = [1,5,1,2,5,5,3,2]
>>> dico_indices(L)
{1: [0, 2], 5: [1, 4, 5], 2: [3, 7], 3: [6]}
```
2. Pour le dictionnaire des occurences, on peut alors utiliser le dictionnaire des indices et calculer les longueurs des listes en valeur.
```
def dico_occur_bis(L):
    D = dico_indices(L)
    D_occ = {}
    for x in D:
        D_occ[x] = len(D[x])
    return D_occ
```
```
>>> L = [1, 5, 1, 2, 5, 5, 3, 2]
>>> dico_occur_bis(L)
{1: 2, 5: 3, 2: 2, 3: 1}
```

Solution (exercice 7) [Énoncé]

1.
```
def plusfrequent(L):
    D = dico_occur(L)
    # on recherche l'effectif le plus important
    k_maxi = L[0] # l'une des clefs de dico_occur
    v_maxi = D[k_maxi]
    for k in D:
        v = D[k]
        if v > v_maxi:
            k_maxi = k
            v_maxi = v
    return k_maxi, v_maxi/len(L)
```
La fonction marche aussi pour des chaînes.
```
>>> c = "les chaussettes de l'archiduchesse sont-elles sèches"
>>> dico_occur(c)
{'l': 4, 'e': 9, 's': 10, ' ': 5, 'c': 4, 'h': 4, 'a': 2, 'u
': 2, 't': 3, 'd': 2, "'": 1, 'r': 1, 'i': 1, 'o': 1, 'n': \
↪ 1, '-': 1, 'è': 1}
>>> L = [1, 1, 2, 3]
>>> plusfrequent(c)
('s', 0.19230769230769232)
>>> plusfrequent(L) # fonctionne encore pour une liste
(1, 0.5)
```
2. Pour la seconde question, on commence donc par générer une urne.
```
import random as rd
candidats = ["CAhyerre", "JHarter", "NLescure", "MJubault", "J
PBellier", "AFernandez"]
urne = []
for _ in range(80):
    urne.append(rd.choice(candidats))
```
Ensuite, on regarde qui est l'élément le plus fréquent :
```
>>> gagnant = plusfrequent(urne)
>>> gagnant
('NLescure', 0.2375)
```
Bravo NLescure ! À chaque exécution de l'éditeur, l'urne sera régénérée et le gagnant modifié.

Solution (exercice 8) [Énoncé]

Pour simplifier, on renvoie les indices des mots hors espaces.
```
def index(c):
    liste_mots = c.split()
    indice = 0 # indice du mot dans le texte
    index = {}
    for mot in liste_mots:
        if mot not in index:
            index[mot] = [indice]
        else:
            index[mot].append(indice)
        indice += len(mot)
    return index
```
```
>>> c = "le temps est beau le ciel est bleu le lycée est beau et
 bleu"
>>> index(c)
```

```
{'le': [0, 14, 27], 'temps': [2], 'est': [7, 20, 34], 'beau': [1
0, 37], 'ciel': [16], 'bleu': [23, 43], 'lycée': [29], 'et': [4
1]}
```