Informatique tronc commun 1re année

Cours à lire pour le TP6 : Dictionnaires, 1re partie

Résumé

Une liste est une façon de construire une collection d'objets mis l'un après l'autre, c'est-à-dire que ses éléments sont indexés par des entiers (leur indice de position) et c'est grâce à cette façon de les classer qu'on peut retrouver facilement un élément d'une liste si on connaît son index (indice).

Un dictionnaire est aussi une façon de construire une collection d'objets mais on les indexe par des *clefs* qui ne sont pas nécessairement des entiers. Connaître la clef d'un élément suffit à le retrouver facilement.

1 Les dictionnaires, présentation

1.1 Généralités

Un dictionnaire est une collection de valeurs quelconques indexées par des clefs (on peut parler de *structure de données associative*).

- Les **valeurs** d'un dictionnaire sont de type quelconque ; pensez aux définitions dans un dictionnaire de langue par exemple.
- Les **clefs** d'un dictionnaire sont des objets de types non mutables (donc ni des listes ni des dictionnaires) ; pensez aux mots à définir, aux entrées dans un dictionnaire de langue.
- Une clef n'apparaît qu'une fois dans un dictionnaire.
- On peut modifier la valeur associée à une clef.
- On peut ajouter une clef ou la supprimer.
- On peut compter les éléments (clefs) d'un dictionnaire.
- On peut parcourir un dictionnaire.

La stratégie de Python pour chercher une clef (et sa valeur associée) dans un dictionnaire sera vue en deuxième année. En attendant il suffira d'admettre que l'on peut retrouver rapidement la valeur associée à la clef dans la mémoire de la machine. C'est un des points essentiels du dictionnaire. Ainsi la recherche d'une clef n'est guère plus difficile que la recherche d'un indice d'une liste.

1.2 Un peu de syntaxe

```
1 dico = {}  # crée un dictionnaire vide
2 dico["prem"] = 2 # crée la clef "prem" et lui associe la valeur 2
3 dico["prem"] = 4  # modifie la valeur associée à la clef "prem"
4 dico["deuze"] = 8  # crée la clef "deuze" et lui associée la valeur 8
5 dico[7] = [2, "g"]  # crée la clef 7 et lui associe la valeur [2, "g"]
```

Après ces lignes de code, `dico` vaut `{"prem": 4, "deuze": 8, 7: [2, "g"]}`

1.3 Parcours de dictionnaire

À titre d'exemple, on utilise une liste de chaînes de caractères pour fabriquer un dictionnaire :

```
dico = {}
for x in ["un", "trois", "vent", "fois", "urne"]:
    dico[x] = x[0]
```

Ainsi, `dico` vaut `{"un": "u", "trois": "t", "vent": "v", "fois": "f", "urne": "u"}`. Notez que les clefs sont bien deux à deux distinctes (mais pas forcément les valeurs).

On peut parcourir le dictionnaire par exemple afin d'en imprimer les valeurs.

```
for clef in dico:
    print(clef, dico[clef])
```

affiche :

```
un u
trois t
vent v
fois f
urne u
```

Attention, on ne peut pas utiliser le slicing pour extraire des sous-dictionnaires.[1]

2 Recherche d'une valeur dans une collection

2.1 Recherche d'une valeur dans une liste ou une chaîne de caractères

Chercher si une valeur est présente dans une collection de type liste ou une chaîne de caractères, dans le cas général, nécessite de parcourir la collection et de regarder chaque élément. Donc, pour une collection de longueur n il faut (dans le pire des cas) effectuer n comparaisons. D'où la fonction :

```
1  def estDans(valeur, collection):
2      """Entrée : une valeur et une collection (liste ou chaîne de caractères)
3         Sortie : True si la valeur est dans la collection, False sinon"""
4      for élément in collection:
5          if élément == valeur:
6              return True
7      return False
8
9  résultat1 = estDans(8, [3, 8, 1])
10 résultat2 = estDans("a", "une phrase")
```

On rappelle que `return` met fin à l'exécution de la fonction, donc si on trouve un élément égal à la valeur, la boucle est interrompue en même temps que la fonction (ligne 5).

Notez que Python fournit une syntaxe pour réaliser la même opération :

```
1  résultat1 = 8 in [3, 8, 1]
2  résultat2 = "a" in "une phrase"
```

sa documentation expliquant que cette syntaxe réalise le même parcours que la fonction `estDans`, et implique donc, au pire, n opérations. Cette syntaxe est donc plus légère, mais pas plus efficace.

2.2 Recherche d'une clef dans un dictionnaire

Pour un dictionnaire, l'opérateur `in` mène la recherche sur les *clefs*. Sur l'exemple du dictionnaire du paragraphe 1.3 :

```
résultat = "trois" in dico
```

Après exécution, `résultat` vaut `True` car la clef `"trois"` est bien présente dans `dico`.
La différence avec les listes ou chaînes de caractères est que cette recherche prend un temps constant, indépendamment de la taille du dictionnaire.

Pour le moment, retenez simplement que chercher une clef dans un dictionnaire est typiquement beaucoup plus rapide que chercher une valeur dans une liste ou une chaîne de caractères.

[1] Principalement parce que les clefs peuvent prendre des valeurs variées : entière, flottante, booléenne, chaîne de caractères, tuple de valeurs... Donc il n'est pas évident de donner un sens au slicing.

Par contre, chercher une *valeur* dans un dictionnaire nécessite de parcourir le dictionnaire et peut donc prendre une durée proportionnelle au nombre d'éléments dans le dictionnaires, comme avec une liste ou une chaîne de caractères. Notez que l'opérateur `in` ne sait pas faire cette recherche, il faut écrire une fonction.

3 Exemple d'usage des dictionnaires : comptage

On souhaite écrire une fonction qui renvoie le nombre d'apparitions de chaque lettre `A`, `C`, `G`, `T` dans une chaîne de caractères constituée uniquement de ces quatre lettres.

Pour cela on va d'abord créer un dictionnaire contenant les clefs `"A"`, `"C"`, `"G"` et `"T"` dont les valeurs sont toutes à 0 (elles serviront de compteurs).

```
1 def initialisation_compteurs():
2     compteur = {}
3     compteur["A"] = 0
4     compteur["C"] = 0
5     compteur["G"] = 0
6     compteur["T"] = 0
7     return compteur
```

Ainsi, cette fonction renvoie le dictionnaire `{"A": 0, "C": 0, "G": 0, "T": 0}`.

Puis on crée une fonction qui parcourt le texte et, pour chaque lettre, incrémente le compteur associée :

```
def compteur_ACGT(texte):
    """Entrée : une chaîne composée des caractères A, C, G, T exclusivement
       Sortie : le dictionnaire dont les clefs sont les lettres
                et les valeurs les compteurs de ces lettres"""
    compteur = initialisation_compteurs()
    for lettre in texte:
        compteur[lettre] += 1
    return compteur
```

Par exemple, Avec `compteur_ACGT("ACCTTGGGGGGTTACA")` renvoie `{"A": 3, "C": 3, "G": 7, "T": 4}`.

Question : commentez l'utilisation d'une liste à la place du dictionnaire pour les compteurs.