Chap 12. Python 3 : Dictionnaires

Livre p 113 – Chap 8 Utilisation avancée des Tableaux
Livre p 189 – Chap 14 *n*-uplets et dictionnaires

1. Ensembles

a. Définition

En Python, le type ensemble (″set″) est une liste non ordonnée d'éléments tous différents, encadrés par des accolades {…, …}.
Attention : La commande a = {} ne définit pas un ensemble vide mais un dictionnaire vide ! (voir 2.)
Pour définir un ensemble vide, utiliser la commande : a = *set*()
Remarque : La commande *len*(ensemble) permet de connaître le nombre d'éléments de l'ensemble mais les éléments n'étant pas ordonnés, on ne peut pas récupérer le $i^{ème}$ élément avec un index !

b. Parcours

On peut parcourir un ensemble avec une boucle : « *for* e *in* ensemble : ».
Tous les éléments seront pris, mais on ne sait pas dans quel ordre !

c. Conversions

On peut convertir une liste ou une chaine de caractères en un ensemble, les doublons sont alors supprimés.

Exemples
set("hello") → {"o", "l", "e", "h"}
set([1, 1, 1, 2, 2, 3]) → {1, 2, 3}

d. Opérations

On peut utiliser certaines opérations ensemblistes…

- en_1 | en_2 : union des ensembles en_1 et en_2.
- en_1 & en_2 : intersection des ensembles en_1 et en_2.
- en_1 – en_2 : différence entre les ensembles en_1 et en_2.
- en_1 ^ en_2 : ou exclusif entre les ensembles en_1 et en_2.
- e *in* en_1 : teste si l'élément e est dans l'ensemble en_1.
- en_1 < en_2 : teste si l'ensemble en_1 est inclus dans l'ensemble en_2.

Exemples
en_1 = {1, 2, 3, 4}
en_2 = {2, 4, 6}
en_1 | en_2 → {1, 2, 3, 4, 6}
en_1 – en_2 → {1, 3}
en_1 ^ en_2 → {1, 3, 6}
3 in en_1 → True
{2, 3} < en_1 → True

e. Méthodes

Les méthodes suivantes modifient l'ensemble sur lequel elles agissent.

- ensemble.*pop*() : renvoie un élément de l'ensemble puis le supprime de l'ensemble.
- ensemble.*add*(e) : ajoute l'élément e à l'ensemble.
- ensemble.*remove*(e) : supprime l'élément e de l'ensemble. (Erreur si l'élément n'y est pas !)
- ensemble.*discard*(e) : supprime l'élément e de l'ensemble. (Pas d'erreur si l'élément n'y est pas.)

2. Dictionnaires

a. Définition

En Python, le type dictionnaire (″dict″) est lui aussi non ordonné et encadré par des accolades mais contient deux informations : les clefs et les valeurs : {clef_1 : valeur_1, clef_2 : valeur_2, …}.
Les clefs des dictionnaires sont comparables aux indices des listes mais peuvent être :

- des nombres. **Exemple** : chaines = {1 : ″TF1″, 2 : ″France 2″, 3 : ″France 3″}
- des chaînes de caractère. **Exemple** : prix = {″stylo″ : 0.95, ″trousse″ : 5.90, ″cahier″ : 1.25}
- des tuples (pas des listes). **Exemple** : points = {(0, 0) : ″O″, (2, 1) : ″A″, (-1, 3) : ″B″}

b. Parcours

On peut parcourir un dictionnaire avec une boucle : « *for* e *in* dico : ».
Par défaut le parcours se fait sur les clefs (méthode dico.*keys*() inutile), mais on peut faire un parcours sur les valeurs avec la méthode dico.*values*(), ou sur les couples (clef, valeur) avec la méthode dico.*items*().

c. Utilisation

Très peu d'opérations fonctionnent sur les dictionnaires !

- *len*(dico) : permet de connaître le nombre de clefs du dictionnaire.
- dico[k] : renvoie la valeur associée à la clef k du dictionnaire.
  Attention : Si la clef n'existe déjà, provoque une erreur !
- dico[k] = v : associe la valeur v à la clef k du dictionnaire.
  Attention : Si la clef existe déjà, la valeur précédente est remplacée par la nouvelle ; sinon, une nouvelle association est créée.
- k in dico : teste si la clef k est dans le dictionnaire. (Ne teste pas si elle est dans les valeurs)
- *del* dico[k] : supprime le la clef k (et la valeur qui lui est associée) dans le dictionnaire.

d. Méthodes

Les méthodes suivantes ne modifient pas le dictionnaire sur lequel elles agissent.

- dico.*copy*() : effectue une copie indépendante du dictionnaire
  Attention : L'écriture : di_2 = di_1 n'effectue pas une copie, il faut écrire : di_2 = di_1.*copy*()
- dico.*get*(k, v_defaut) : si la clef k est présente, renvoie la valeur associée cette clef ; sinon, renvoi la valeur v_defaut (*none* si rien n'est précisé)

Les méthodes suivantes modifient le dictionnaire sur lequel elles agissent.

- dico.*pop*(k) : renvoie la valeur associée à la clef k puis supprime cette clef du dictionnaire.
- di_1.*update*(di_2) : met à jour le dictionnaire di_1 avec toutes les clefs du dictionnaire di_2.
  Attention : si une des clefs existe déjà, la valeur précédente est remplacée par la nouvelle ; sinon, une nouvelle association est créée.

3. Modélisation d'un tableau

a. Par des listes

En Python, une liste (ou un tuple) peut contenir d'autres listes (ou d'autres tuples), cela permet de créer des tableaux à plusieurs dimensions.

1	2	3	4
5	6	7	8
9	10	12	13

Exemple : Le tableau ci-contre peut être modélisée par la liste :
tableau = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12]].
tableau[i] représente la $(i + 1)^{ème}$ ligne.
tableau[i][j] correspond à la valeur située à la $(i + 1)^{ème}$ ligne et la $(j + 1)^{ème}$ colonne.

b. Par des dictionnaires

Lorsque les colonnes d'un tableau ont un titre, il peut être plus simple d'utiliser des listes de dictionnaires plutôt que des listes de listes, les clefs correspondent alors aux titres des colonnes.

prénom	nom	âge
Pierre	Machin	16
Paul	Truc	
Jack	Ouille	17

Exemple : Le tableau ci-contre peut être modélisée par la liste :
tableau = [{″prénom″ : ″Pierre″, ″nom″ : ″Machin″, ″âge″ : ″16″},
{″prénom″ : ″Paul″, ″nom″ : ″Truc″, ″âge″ : ″″},
{″prénom″ : ″Jack″, ″nom″ : ″Ouille″, ″âge″ : ″17″}]
tableau[i] représente la $(i + 1)^{ème}$ ligne de valeurs.
tableau[i][clef] correspond à la valeur située à la $(i + 1)^{ème}$ ligne et à la colonne nommée clef.

c. Le module numpy

Pour les tableaux à deux dimensions ne comportant que des nombres, il existe un objet mathématique très pratique : les matrices. Le module *numpy* (*np* en abrégé) permet de les manipuler facilement.