Dictionnaires - Résumé

Structure de dictionnaire

Une **liste** L, un **tuple** t, un **tableau numpy** a (array) sont des structures de données dont ***les éléments sont indexés par des entiers*** : on accède à l'élément d'index i par `L[i]`, `t[i]`, `a[i]`.

Un dictionnaire (ou encore table d'association) est une structure de données associant un ensemble de clefs (keys) à un ensemble de valeurs (values).
Dans la suite, C désigne l'ensemble des clefs et V l'ensemble des valeurs.
Les clés peuvent être de type str, int, float, tuple, bool
Etant donnés un ensemble C de clés possibles et un ensemble V de valeurs possibles, un dictionnaire représente un ensemble fini d d'associations (clé, valeur) (d est un sous-ensemble de C x V) tel qu'à chaque clé corresponde au plus une valeur : si (k1, v1) et (k2, v2) sont deux associations du dictionnaire d et k1 = k2, alors v1 = v2.
Si k est une clé du dictionnaire d, la seule valeur qui lui est associée sera notée `d[k]`.
On note $\{k_0 : v_0,\ k_1 : v_1, ..., k_{n-1} : v_{n-1}\}$ le dictionnaire de n objets qui associe la valeur v_i à la clé k_i avec $0 \leq i \leq n-1$.
Un dictionnaire d est donc une structure de données dont les éléments sont indexés par des clés : on accède à la valeur v associée à la clé c par **d[c]** (i.e. d[c] renvoie la valeur v).
Un dictionnaire est une structure de données **mutable** (modifiable) et **non ordonnée** (on ne pas trier un dictionnaire).

Exemple : tableau des coefficients à CCINP-physique

```
ccinp_physique = {'Maths':14, 'phys':15, 'chimie':8, 'philo':9, 'mod':8, 'lva':4, 'lvb':2}
```

Exemple d'application réelle

La gestion des **noms de domaines** (DNS, Domain Name System) utilise un dictionnaire : à chaque adresse IP (Internet Protocol) de la forme "https://fr.wikipedia.org/wiki/Adresse_IP" (chaîne de caractères = clé) correspond une adresse de la forme 135.160.254.100 (valeur).

Opérations sur les dictionnaires

- Création d'un dictionnaire
- Existence d'une clé c dans le dictionnaire
- Lecture de la valeur v associée à une clé existante c / parcours du dictionnaire
- Ajout d'une nouvelle association clé/valeur
- Suppression d'une association clé/valeur

Ensemble vs dictionnaire

Ne pas confondre ensemble et dictionnaire (accolades dans les deux cas).

```
# Ensemble
e = {'ab', 1, -2.36, True}
type(e))          # Renvoie :

# Dictionnaire
d = {'a1':4, 'azerty':'fr', 'No':False}
type(d)          # Renvoie :
```

Manipulation des dictionnaires

Création d'un dictionnaire

Méthode 1 par extension

```
nom_variable = {}          # Crée un dictionnaire vide
nom_variable = {cle1:valeur1, cle2: valeur2, etc}
```

Méthode 2 conversion (dict)

```
nom_variable = dict(iterable)
```

où *iterable* est une liste, un tuple,... de couples de la forme (clé, valeur).

Méthode 3 par compréhension

Syntaxe courte :

```
nom_variable = {formule for variable in range(...)}
```

Ou bien création d'un dictionnaire vide puis remplissage à l'aide d'une boucle :

```
nom_variable = {}
for ... in range(...):
    nom_variable[...] = ...
```

Accès à un élément

```
nom_du_dictionnaire[cle]
```

Afin d'éviter l'interruption "KeyError", on peut **tester la présence d'une clé dans le dictionnaire** :

```
cle in nom_du_dictionnaire
```

La complexité temporelle de ce test d'appartenance est O(1).

Parcours d'un dictionnaire

Méthodes associées aux dictionnaires

Un dictionnaire est un objet itérable (comme les ranges, les listes, les tuples, les chaines de caractères).
Si d est un dictionnaire, une boucle for et les méthodes d.keys(), d.values() et d.items() permettent de parcourir le dictionnaire d.

```
# Parcours par clé
for c in nom_du_dictionnaire.keys():
for c in nom_du_dictionnaire:      # Raccourci
# Parcours par valeur
for v in nom_du_dictionnaire.values():
# Parcours par couple(clé, valeur)
for c, v in nom_du_dictionnaire.items()
```

Les noms de variable c,) sont arbitraires.

Noter que les résultats de ces commandes ne sont pas des listes (utiliser la fonction type() pour s'en convaincre).
Si nécessaire, on peut convertir ces structures en listes en utilisant la fonction **list()**.

Ajout, suppression dans un dictionnaire

Ajout d'un seul élément - Méthode 1

```
nom_du_dictionnaire[cle] = valeur
```

⚠ Remplace (écrase) la valeur si la clé est déjà existante.

Ajout de plusieurs éléments - Méthode 2

```
nom_du_dictionnaire.update({cle1: valeur1, cle2: valeur2,...})
```

Suppression

```
del nom_du_dictionnaire[cle]
```

⚠ Génère une erreur si la clé n'existe pas.

Copie d'un dictionnaire

```
d2 = d1.copy()      # Même instruction que pour les listes}
```

Valeurs d'un dictionnaire

Tout objet Python (nombre, chaîne, liste, dictionnaire...) peut être la **valeur** associée à une clé.

Clés d'un dictionnaire

Objet mutable (i.e. modifiable) vs objet non nutable

On distingue les objets mutables i.e. **modifiables** (liste, tableau numpy, ensemble, dictionnaire) des objets non mutables (entier, flottant, booléen, chaîne, tuple).

Les commandes suivantes pemettent de vérifier si un objet est mutable ou non.

Instruction	Exécution
3 = 8	SyntaxError : cannot assign to literal
1.0 = 3.14	SyntaxError : cannot assign to literal
True = False	SyntaxError : cannot assign to True
`s = 'abcde' s[2]= 'z'`	TypeError : 'str' object does not support item assignment
`t = (0, 1, 2, 3) t[2]= 18`	TypeError : 'tuple' object does not support item assignment
`L = [0, 1, 2, 3] ; L[2]= 3.14`	[0, 1, 3.14, 3]
`d ={0: 'faux', 1: 'fake'}; d[1]= 'vrai'`	{0 : 'faux', 1 : 'vrai'}

Clés d'un dictionnaire

Les **clés** d'un dictionnaire peuvent être tout objet de type non mutable récursivement (récursivement signifie que si une clé est un tuple par exemple, les éléments du tuple doivent être non mutables).
Une liste, un dictionnaire (objets mutables) ne peuvent pas être des clés d'un dictionnaire.