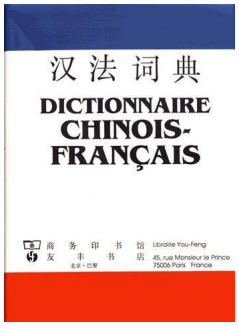
TP-Cours – Les dictionnaires et les ensembles

Objectifs :

- ⇨ Comprendre l'utilité des dictionnaires et des ensembles
- ⇨ Savoir utiliser les dictionnaires et les ensembles de python



I - Les dictionnaires

1) Définition

Un dictionnaire (appelé aussi **tableau associatif** ou table d'association) est un type de données associant à un ensemble de clefs un ensemble correspondant de valeurs. Chaque clef est associée à une valeur.

Dans l'exemple des données exifs, on pourrait créer un dictionnaire dont les clés seraient les différentes méta-données.
Voyons comment cela se passe en python en utilisant la console pour faire quelques essais.

Clé	Valeur
Nom_fichier	PIC01245653.jpg
Date_heure	11/06/2013 14:01:13
Marque	SAMSUNG
Modèle	GT-I9300
ISO	80
Dimension_X	3264
Dimension_Y	2448
Orientation	En haut à gauche

Création d'un dictionnaire :

```
>>> d = {'un' : 1, 'deux' : 2, 'trois' : 3, 'quatre' : 4}
>>> d
{'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4}
>>> type(d)
Dict
>>> a = {}     # crée un dictionnaire vide
```

On note l'utilisation d'accolades pour les dictionnaires, là où on avait des parenthèses pour les tuples ou des crochets pour les listes. Les clés et les valeurs sont séparés par le signe « : » et les couples « clé : valeur » (on parle d'*éléments* ou d'*entrées de dictionnaire*) sont séparés par des virgules (comme pour les listes et les tuples).

On peut également créer un dictionnaire à partir d'une liste avec l'instruction `dict()` ou à l'aide d'une compréhension comme pour les listes :

```
>>> liste = [(1, 2), (3,True), (5, 'abc')]
>>> d = dict(liste)
>>> d
{1: 2, 3: True, 5: 'abc'}
>>> d2 = {cle : cle**2 for cle in range(5)}
{0: 0, 1: 1, 2: 4, 3: 9, 4: 16}
```

On peut accéder à un élément un peu à la manière des listes :

```
>>> d = {'un': 1, 'deux': 2, 'trois': 3, 'quatre': 4}
>>> d['deux']
2
>>> x = 'trois'
>>> d[x] = 'Allier'
>>> d[1.5] = True
>>> d[-42] = [1, 2, 3]
>>> d
{'un': 1, 'deux': 2, 'trois': 'Allier', 'quatre': 4, 1.5: True, -42: [1, 2, 3]}
>>> d['zero']
Traceback (most recent call last):
  File "<pyshell>", line 1, in <module>
KeyError: 'zero'
```

Au passage on remarque :

- Que les éléments sont affichés dans l'ordre d'ajout au dictionnaire. En fait un dictionnaire n'a pas d'ordre contrairement aux listes.
- Que les clés[1] et les valeurs peuvent être de n'importe quel type et qu'on peut mixer les types comme dans les listes.
- Un dictionnaire ne peut pas contenir des clés dupliquées. Lorsqu'on a fait « `d[x] = 'Allier'` », on a changé la valeur de cette clé, mais pas crée une nouvelle entrée de dictionnaire.
- La référence à une clé inexistante provoque une erreur (levée d'exception « `KeyError` »).

Un dictionnaire est une structure de donnée *itérable*. On peut donc utiliser une boucle for pour parcourir tous ses éléments :

```
>>> for element in d:
        print(element)
un
deux
trois
quatre
1.5
-42
```

On remarque que la console nous a affiché uniquement les clés et non les valeurs. Pour obtenir uniquement les valeurs, uniquement les clés ou les deux, il faut utiliser les mots-clés suivants :

Avec la méthode `values()`	Avec la méthode `keys()`	Avec la méthode `items()`

`for valeur in d.values():`
`  print(valeur)`
`1`
`2`
`Allier`
`4`
`True`
`[1, 2, 3]`
	`for valeur in d.keys():`
	`  print(valeur)`
	`un`
	`deux`
	`trois`
	`quatre`
	`1.5`
	`-42`
		`for valeur in d.items():`
		`  print(valeur)`
		`('un', 1)`
		`('deux', 2)`
		`('trois', 'Allier')`
		`('quatre', 4)`
		`(1.5, True)`
		`(-42, [1, 2, 3])`

2) Les méthodes des dictionnaires

La classe Python pour les dictionnaires est `dict`**.** La liste suivante énumère les méthodes qui peuvent être appelées à partir d'un objet dictionnaire.

- `keys()` : Retourne une séquence de toutes les clés.
- `values()` : Retourne une séquence de toutes les valeurs.
- `items()` : Retourne une séquence de tous les éléments sous forme de tuples. Chaque tuple est sous la forme (clé, valeur).
- `clear()` : Supprime tous les éléments.
- `get(clé)` **:** Renvoie la valeur de la clé.
- `pop(clé)` **:** Supprime l'élément dont on donne la clé et retourne sa valeur.
- `popitem()` **:** Renvoie une paire clé/valeur sélectionnée au hasard sous forme d'un tuple et supprime l'élément sélectionné.

La méthode `get(clé)` est similaire à `NomDictionnaire[clé]`, sauf que la méthode `get()` retourne `None` si la clé est pas dans le dictionnaire plutôt que de déclencher une exception.

[1] En fait les clés doivent être d'un type non mutable (`int`, `str`, `float`, `bool` conviennent, mais les listes ne conviennent pas)

3) Revenons au problème du codage

Question 1 :
En utilisant une structure de type dictionnaire, écrire une deuxième fonction de codage `codageDict(texte)` ayant la même fonction que la précédente (voir activité préparatoire).
On pourra convertir le tableau de tuple (lettre, code) en dictionnaire simplement avec l'instruction `dict(code)`.

Question 2 :
En utilisant le module `timeit`, comparer les temps d'exécution des deux fonctions (avec et sans utilisation d'un dictionnaire). Expliquer la différence.

Aide :
Pour évaluer le temps d'exécution d'un bout de code, on peut utiliser l'instruction :

```
import timeit
print(timeit.timeit('instruction à mesurer',number=1,
globals=globals()))
```

`number` est le nombre d'exécution. On peut l'augmenter pour améliorer la précision de la mesure.

Explication technique :
Un dictionnaire utilise un mécanisme appelé « table de hachage » (*hash table*) qui permet de retrouver directement l'élément du dictionnaire en calculant juste une fonction de hachage sur la clé. On accède ainsi aux éléments quasiment en temps constant (c'est-à-dire que le temps d'accès à un élément est le même où qu'il soit dans le dictionnaire et indépendamment du nombre d'élément dans le dictionnaire.

II – Les ensembles

1) Qu'est-ce qu'un ensemble ?

Les ensembles (ou « sets » en anglais) sont des collections non ordonnées d'éléments uniques. Elles permettent de faire des tests d'appartenance très rapides et supportent les opérations mathématiques courantes sur les ensembles.

Remarques :

- **Un ensemble ne peut pas contenir de doublons**
- Un ensemble est mutable (il y a les `frozensets` qui sont des ensembles non mutables que l'on ne détaillera pas ici)
- Les ensembles sont notés comme les dictionnaires entre accolade. En fait c'est comme si c'étaient des dictionnaires contenant uniquement des clés et pas de valeurs.
- Les ensembles sont des objets *itérables* que l'on peut donc parcourir avec une boucle for.

Création d'un ensemble :

```
>>> e = {1, 2, 18, 52, 'abc', 3.14}
{1, 18, 2, 3.14, 52, 'abc'}
>>> type(e)
set
>>> e.add(15)
>>> e.add((12,'xy'))
>>> e
{(12, 'xy'), 1, 15, 18, 2, 3.14, 52, 'abc'}
>>> e.add([3,8])
Traceback (most recent call last):
  File "<ipython-input-158-8b1ae03fd228>", line 1, in <module>
```

```
    e.add([3,8])
TypeError: unhashable type: 'list'
```

On remarque :

- Qu'il n'y a pas d'ordre dans les éléments. Ou plus exactement python suit sa logique pour classer les éléments qui n'est pas simple (pas d'ordre numérique notamment). En particulier l'ordre dans lequel les éléments sont mis dans l'ensemble n'a aucune influence sur leur placement.
- Qu'on ne peut ajouter que les types non-mutables[2] (int, str, float, tuples de ces types de base), mais pas d'objets mutables (pas de listes entre autre).
- Que pour créer un ensemble vide, on ne peut pas utiliser simplement `{}` car il s'agit déjà du dictionnaire vide. On devra donc utiliser `set()` pour créer un ensemble vide.

2) Les méthodes des ensembles

La classe Python pour les ensembles est `set`**.** La liste suivante énumère les méthodes qui peuvent être appelées à partir d'un objet ensemble.

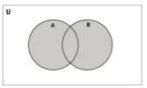
- `add(x)` : Ajoute l'élément `x` à l'ensemble (il doit être de type non mutable).
- `remove(y)` : Retire l'élément `y` de l'ensemble (si y n'est pas dans l'ensemble il y a une exception).
- `discard(y)` : Retire l'élément `y` de l'ensemble (si y n'est pas dans l'ensemble il ne se passe rien).
- `clear()` : Supprime tous les éléments.
- `pop()` **:** Supprime un élément aléatoire de l'ensemble et renvoie sa valeur
- `update(L)` **:** Ajoute les éléments de l'itérable `L` à l'ensemble (on peut ajouter une liste ou un autre ensemble par exemple). Les doublons sont éliminés.

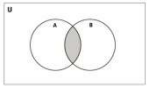
3) Les opérations sur les ensembles

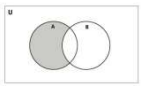
Les ensembles ont la particularité de supporter les opérations classiques de math sans avoir à les programmer :

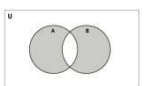
Application 1 :
1) Compléter la dernière colonne du tableau suivant (essayer de deviner puis testez avec python)

Opération	Syntaxe	Exemple avec
		`e1 = {1, 2, 3, 4, 5}`
		`e2 = {4, 5, 6, 7, 8}`

 Union	`e = e1 | e2`
	ou
	`e = e1.union(e2)`	

 Intersection	`e = e1 & e2`
	ou
	`e = e1.intersection(e2)`	

 Différence	`e = e1 - e2`
	ou
	`e = e1.difference(e2)`	

 Différence symétrique	`e = e1 ^ e2`
	ou
	`e1.symmetric_difference(e2)`	

[2] Comme nous le dit le message d'erreur, il faut que les types soient « *hachables* » et donc non modifiables (sinon leur signature (*hash*) changerait lorsque leur valeur change.

2) Même travail avec ce deuxième tableau

Type de comparaison	Syntaxe	Exemple avec
		`e1 = {1, 2, 3, 4, 5}`
		`e2 = {4, 5, 6, 7, 8}`
		`e3 = {2, 3}`

Disjoint Les deux ensembles n'ont aucun élément en commun	`e1.isdisjoint(e2)`	`e1.isdisjoint(e2) vaut .....` `e2.isdisjoint(e3) vaut ......`
Inclusion Les éléments du premier ensemble appartiennent tous au deuxième	`e1.issubset(e2)` `e1 <= e2`	`e1.issubset(e2) vaut .......` `e3.issubset(e1) vaut .......`
Inclusion stricte chaque élément de e1 est également dans e2, et au moins un élément de e2 n'est pas dans e1	`e1 < e2`	`e3 < e1 vaut .......` `e2 < e2 vaut .......`

TRAVAIL A FAIRE POUR LA PROCHAINE SEANCE :
Compléter le résumé de cours suivant :

Les dictionnaires sont .....

.....

.....

Ils permettent de .....

.....

.....

Les ensembles sont .....

.....

.....

Ils permettent de .....

.....

.....