Dictionnaires

Définition

Un dictionnaire est une collection non ordonnée de paires : (Clé : valeur)

Les clés

Les clés doivent être immuables. Elles doivent aussi être uniques. Et ces clés doivent être « hashable ». Cela implique par exemple que les clés ne peuvent pas être des listes ou tableaux Numpy.

Si la clé est une variable, alors la clé sera la valeur de cette variable au moment de la création de la de la paire (clé : valeur). Une modification ultérieure de la variable ne modifiera pas la clé du dictionnaire.

Les valeurs

Les valeurs peuvent être de tout type (y compris des listes). Elles peuvent être changées.

Si la valeur est une variable. Alors la valeur du dictionnaire aura la valeur de la variable. Une modification ultérieure de la variable ne modifiera pas la valeur du dictionnaire.

Opérations de base sur les dictionnaires

Création d'un dictionnaire vide : Dico={}

```
>>> Dico={}                    ou          >>> Dico=dict()
>>> Dico
---renvoie--> {}
```

Création d'un dictionnaire avec des paires (clé : valeur) :

```
>>> Dico={1 : 1 , 2 : 3 , 'a' : 7 , (45,'b') : 4}
>>> Dico
---renvoie--> {1: 1, 2: 3, 'a': 7, (45, 'b'): 4}
```

Ajout d'une paire dans un dictionnaire

```
>>> Dico['cle1']='valeur1'
>>> Dico
---renvoie--> {1: 1, 2: 3, 'a': 7, (45, 'b'): 4, 'cle1': 'valeur1'}
```

Suppression d'une paire dans un dictionnaire

Première solution

```
>>> del Dico['cle1']
>>> Dico
---renvoie--> {1: 1, 2: 3, 'a': 7, (45, 'b'): 4}
```

Deuxième solution

Supprime la cle et la valeur et renvoie la valeur qui peut alors être affectée à une variable.

```
>>> Dico.pop('a')
---renvoie--> 7
>>> variable=Dico.pop((45, 'b'))
>>> variable
---renvoie--> 4
>>> Dico
---renvoie--> {1: 1, 2: 3}
```

Modification d'une valeur (correspondant à une clé)

```
>>> Dico[1]=2
>>> Dico
renvoie> {1: 2, 2: 3}
>>> Dico[2]+=1
>>> Dico
renvoie> {1: 2, 2: 4}
```

Récupération d'une valeur (correspondant à une clé)

Première solution

```
>>> variable=Dico[2]
>>> variable
renvoie> 4
>>> Dico['clé_inex']
renvoie>
Traceback (most recent call last):
  File "<console>", line 1, in <module>
KeyError: 'clé_inex'
```

Deuxième solution

```
>>> Dico.get(2)
renvoie> 4
>>> Dico.get(3)          # (ne renvoie rien car la clé 3 n'existe pas. Cela ne renvoie pas d'erreur)
>>> Dico.get(3,'Défaut')    #(Si le premier argument de la méthode n'est pas une clé existante du
renvoie> 'Défaut'           #  dictionnaire alors la fonction renvoie le deuxième)
```

Parcours d'un dictionnaire

Un dictionnaire se parcourt par ses clés

```
>>> D={'a' : 1 , 'b' : 2 , 'c' : 3}
>>> for c in D :
>>>     print('Clé :',c,'Valeur :',D[c])
renvoie> Clé : 'a' Valeur : 1
         Clé : 'b' Valeur : 2
         Clé : 'c' Valeur : 3
```

Deux méthodes pour récupérer les clés/valeurs d'un dictionnaire

Il existe deux méthodes « .keys() » et « .values() » retournant l'énumération des clés ou des valeurs.

```
>>> D.keys()
renvoie> dict_keys(['a', 'b', 'c'])
>>> D.values()
renvoie> dict_values([1, 2, 3])
```

Attention le retour des méthodes " .keys() » et « .values() » ne sont pas de type liste.

```
>>> LK , LV = D.keys() , D.values()
>>> type(LK),type(LV)
renvoie> (<class 'dict_keys'>,<class 'dict_values'>)
```

Ces types sont immuables. Ils ne peuvent pas être modifiés ni un élément appelé par un indice.

```
>>> LK[0]
renvoie> Traceback (most recent call last):
           File "<console>", line 1, in <module>
         TypeError: 'dict_keys' object is not subscriptable
```

En revanche, ils peuvent être parcourus ce qui permet un parcourt du dictionnaire par les valeurs

```
>>> for i in D.keys():
>>>     print(i)
renvoie> a
         b
         c
>>> for i in D.values():
>>>     print(i)
renvoie> 1
         2
         3
```

Autres fonction ou méthode s'appliquant aux dictionnaires

La fonction « len » s'applique de la même manière aux dictionnaires qu'aux listes.

```
>>> len(D)
renvoie> 3
```

La fonction « copy » permet de faire une copy du dictionnaire comme pour les listes.

```
>>> DCop=D
>>> DCop['a']=5
>>> DCop
renvoie> {'a': 5, 'b': 2, 'c': 3}
>>> D
renvoie> {'a': 5, 'b': 2, 'c': 3}
>>> DCop=D.copy()
>>> DCop['a']=1
>>> DCop
renvoie> {'a': 1, 'b': 2, 'c': 3}
>>> D
renvoie> {'a': 5, 'b': 2, 'c': 3}
```

Mais comme pour les types 'list' la méthode copy ne fait de copie récursive

Pour une copie récursive, il faut la fonction « deepcopy(D) » du module copy.

```
>>> D={'a': {'cle':'valeur'}, 'b': 2, 'c': 3}
>>> DCop=D.copy()
>>> DCop['a']['cle']='new_val'
>>> DCop
renvoie> ({'a': {'cle': 'new_val'}, 'b': 2, 'c': 3}
>>> D
renvoie> {'a': {'cle': 'new_val'}, 'b': 2, 'c': 3})

>>> from copy import deepcopy
>>> DCop=deepcopy(D)
>>> DCop['a']['cle']='new_new_val'
>>> DCop
renvoie> ({'a': {'cle': 'new_new_val'}, 'b': 2, 'c': 3}
>>> D
renvoie> {'a': {'cle': 'new_val'}, 'b': 2, 'c': 3})
```

La fonction « in » pour la recherche d'un élément dans le dictionnaire

Cette fonction renvoie un booléen si l'élément est dans **les clés** du dictionnaire

```
>>> D={'a' : 1 , 'b' : 2 , 'c' : 3}
>>> 'a' in D :
renvoie> True
>>> 1 in D :
renvoie> False
```

Pour tester si un élément est dans les valeurs du dictionnaire on utilise la méthode « .values() »

```
>>> 1 in D.values() :
renvoie> True
```

Limites et puissances du type dictionnaire

Limite du dictionnaire

Le langage Python impose une restriction sur les clés d'un dictionnaire : certains types de données ne sont pas autorisés. Comme par exemple le type « list ».

```
>>> D={[14,18]:'grande guerre'}
renvoie> Traceback (most recent call last):
           File "<console>", line 1, in <module>
         TypeError: unhashable type: 'list'
```

En revanche on peut utiliser le type **« tuple ».**

```
>>> D={(14,18) :'grande guerre'}
>>> D
renvoie> {(14, 18): 'grande guerre'}
```

En fait les clés doivent être des types immuables (non mutable). Ce qui exclue les types « list ».

Puissance du dictionnaire

Pour qu'un dictionnaire soit efficace, il faut pouvoir accéder rapidement à la valeur associée à une clé. Lors de la recherche de « Dico[clé] » Python ne parcourt pas tout le dictionnaire.

Méthode qui serait en complexité O(n).

Pour cette recherche Python utilise une fonction de « hashage » qui à un élément associe un entier (de type « int »). Cela lui permet d'accéder aux clés qui sont réparties en mémoire par « paquets ».

Méthode qui permet une complexité en O(1). Donc indépendante de la taille du dictionnaire.

La fonction de hashage « hash() » associe à un objet Python un entier (de type « int »)

```
>>> hash(1914)          # A un entier de petite taille la fonction hash() retourne les même entier
renvoie> 1914

>>> hash([14,18])       # A une liste la fonction hash() retourne une erreur
renvoie> Traceback (most recent call last):
           File "<console>", line 1, in <module>
         TypeError: unhashable type: 'list'

>>> hash((14,18))       # A un tuple la fonction hash() retourne une entier
renvoie> 7066744301585418553

>>> hash(3.14159)       # A un réel la fonction hash() retourne une entier
renvoie> 326484311674566659
```

Attention ! La fonction de hashage n'est pas injective.

```
>>> hash(326484311674566659)
renvoie> 326484311674566659          # L'entier est le même que pour 3.14159
```

Cela crée des conflits pour l'affectation des valeurs en mémoire. Conflits qui sont résolus par des méthodes qui ne sont pas au programme de CPGE scientifique.

Exercices

Exercice 1

a- Ecrire une fonction « inverse_dictionnaire(D) » qui prend en argument un dictionnaire D et retourne le dictionnaire inversé où les clés sont les valeurs de D et les valeurs sont les clés correspondantes de D. Cette fonction peut-elle s'appliquer à tous les dictionnaires ? Justifier

b- Ecrire une fonction « intersect_dictionnaires(D1,D2) » qui prend en argument deux dictionnaires D1 et D2 et qui retourne un dictionnaire comportant toutes les clés présentes dans les deux dictionnaires D1 et D2 et dont les valeurs correspondantes sont le couple des valeurs de D1 et D2

c- Soit les deux dictionnaires :

Dico1 = {'C': 6, 'O': 8, 'H': 1, 'Fe': 26, 'Cr': 24, 'Ni': 28, 'Cu': 29, 'Al': 13, 'Zn': 30}

Dico2 = {'carbone': 'C', 'hydrogène': 'H', 'helium': 'He', 'oxygène': 'O', 'zinc': 'Zn' }

Ecrire une fonction « chimie(D1,D2) » qui en utilisant les dictionnaires D1 et D2 et les fonctions précédentes retourne une liste de couples constitués des noms des éléments et de leur numéro atomique.

Exercice 2

Les distances en mathématique correspondent à une définition qui est la suivante. C'est une application f : $E \times E \rightarrow \mathbb{R}^+$ qui pour tout triplet $(x,y,z) \in E^3$ on a les propriétés suivantes :

$f(x,y) \geq 0$ $f(x,y) = 0 \Leftrightarrow x=y$ $f(x,y) = f(y,x)$ $f(x,y) \leq f(x,z) + f(z,y)$

Certaines sont bien connues : distances Euclidiennes, de Manhattan, de Minkowski, de Tchebychev. Ce sont des distances de $\mathbb{R}^n \rightarrow \mathbb{R}^+$ Mais quant est-il des distances d'édition entre deux chaines de caractères ? Pourtant bien utile pour la détection des fautes de frappes ou la correction orthographique.

Une première idée pour définir la distance entre deux mots est de compter le nombre de lettres ajoutées et retirées (indépendamment de leurs positions) et de les additionner pour définir la distance d'édition. Nous allons écrire un code qui fait ce calcul

a- Ecrire une fonction **`dico_lettres(texte)`** qui prend en argument un texte (chaine de caractères) et qui retourne un dictionnaire dont les clés sont les caractères du texte et les valeurs le nombre d'occurrence de chaque caractère dans le texte.

b- Ecrire une fonction **`nbr_sup(mot1,mot2)`** qui prend en argument deux textes (chaine de caractères) et qui retourne le nombre de lettre supprimées pour passer du mot1 au mot 2. Vous utiliserez la fonction **`dico_lettres(texte)`**.

c- Ecrire une fonction **`nbr_ajou(mot1,mot2)`** qui prend en argument deux textes (chaine de caractères) et qui retourne le nombre de lettre ajoutées pour passer du mot1 au mot 2. Vous utiliserez la fonction **`dico_lettres(texte)`**.

d- Ecrire une fonction **`dist_edition(mot1,mot2)`** qui prend en argument deux textes (chaine de caractères) et qui retourne le nombre de lettre ajoutées et retirées (indépendamment de leurs positions) pour passer du mot1 au mot 2. Vous utiliserez les fonctions précédentes.

e- Essayez avec 'chien' et 'niche' Cette distance d'édition est-elle vraiment une distance ?

Exercice 3

Un graphe est décrit par sa matrice d'adjacence (Donnée sous la forme d'une liste de liste ou un tableau numpy à deux dimensions). Les sommets sont les indices des ces listes et les valeurs (liste[i][j] ou tableau[i,j]) sont égales à 1 si les sommets i et j sont voisins ou à 0 sinon.

a- Ecrire une fonction « graphe_mat_dict(M) » qui prend ce graphe sous forme de matrice et retourne ce même graphe sous forme de dictionnaire où les clés sont les sommets et les valeurs les listes des sommets voisins.

b- Ecrire une fonction « Graphe_dict_mat(D) » réalisant l'opération inverse de la fonction précédente.