Chap 5

Les dictionnaires et les ensembles

Enseignante: Mme Lamia MANSOURI

1. Les tableaux associatifs

Un tableau associatif est un type de données permettant de stocker des couples (**cle : valeur**), avec un accès très rapide à la valeur à partir de la clé.
La clé ne peut être présente qu'une seule fois dans le tableau.

Il possède les caractéristiques suivantes :

- L'opérateur d'appartenance d'une clé (**in**)
- La fonction taille (**len()**) donnant le nombre de couples stockés;
- Il est itérable (on peut le parcourir) mais il n'est pas ordonné.

Python propose le type standard **dict**.

2. Les dictionnaires (dict)

Les chaînes, les listes et les tuples: - des suites ordonnées d'éléments.
- On peut accéder à un élément quelconque à l'aide d'un index

- Un dictionnaire en python est une sorte de liste mais au lieu d'utiliser des index , on utilise des clés alphanumériques.
- Les dictionnaires mappent * clés * en * valeurs * .
- Comme les listes, les dictionnaires sont modifiables.
- La **clé** qui peut être n'importe quel type **non-modifiable** (les chaînes,les nombres et les tuples s'ils ne contiennent que des éléments non modifiables)

Syntaxe

```
Nom-Collection= {
Cle1 : valeur1,
cle2 :valeur2,
…
Clen :valeurn
}
```

Pour accéder à la valeur du dictionnaire, nous utilisons la syntaxe suivante:

nom_dictionnaire[clé]

Exemple

4

On veut créer un dictionnaire de langue, pour la traduction de termes informatiques anglais en français.

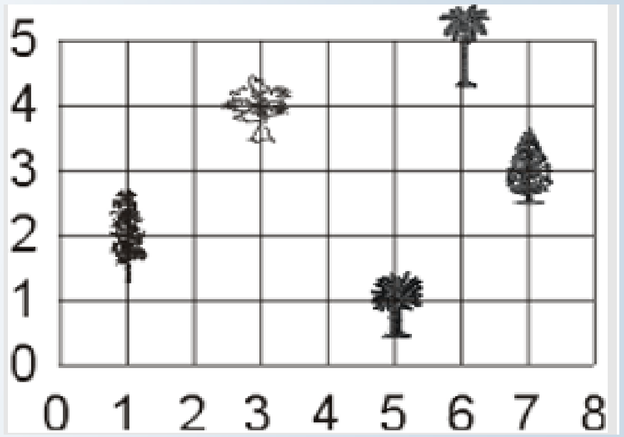
```
dico = {} # création d'un dictionnaire vide
dico['computer'] = 'ordinateur'
dico['mouse'] ='souris'
dico['keyboard'] ='clavier'
print(dico)
#on aura ceci :
# on aura ceci :
# {'computer': 'ordinateur', 'mouse': 'souris', 'keyboard': 'clavier'}

print (dico['mouse'])        # affiche souris
```

Autres exemples de clé

5

Considérons par exemple que nous voulions répertorier des sur un grand terrain rectangulaire. Nous pouvons pour cela utiliser un dictionnaire, dont les clés seront des tuples indiquant les coordonnées x,y de chaque arbre :

```
arb = {}
arb[(1,2)] = 'Peuplier'
arb[(3,4)] = 'Platane'
arb[6,5] = 'Palmier'# on peut éliminer les ()
arb[5,1] = 'Cycas'
arb[7,3] = 'Sapin'
print(arb[(6,5)])# affiche palmier
print(arb)
#on aura cet affichage :
#{(3, 4):'Platane',(6, 5):'Palmier',(5, 1):'Cycas',(1, 2):'Peuplier',(7, 3):'Sapin'}
print(arb[1,2])# affiche Peuplier
print(arb[2,1]) # affiche  ***** Erreur : KeyError: (2, 1)  *****
```



Pour résoudre cet erreur , on utilise la méthode **get().** Le 1er argument transmis à cette méthode est la clé de recherche, 2ème argument est la valeur que nous voulons obtenir en retour si la clé n'existe pas dans le dictionnaire :

```
print(arb.get((1,2), 'néant'))     #  affiche Peuplier
print(arb.get((2,1), 'néant'))     # affiche néant
```

Remarque

- Les dictionnaires ne sont pas des séquences : Les éléments d'un dictionnaire ne sont pas disposés dans un ordre particulier. Des opérations comme **la concaténation et l'extraction** (d'un groupe d'éléments contigus) ne peuvent donc tout simplement pas s'appliquer ici.

Si vous essayez tout de même, Python lèvera une erreur lors de l'exécution du code :

```
print (arb[1:3])
```

```
***** Erreur : KeyError: slice(1, 3, None) *****
```

2.1 Opérations sur les dictionnaires

- **del**

Pour enlever un élément du dictionnaire on utiliser l'instruction del

Exemple :

```
invent = {'pommes':430,'bananes': 312,'oranges':274,'poires': 137}
print(invent)
# affiche {'oranges': 274, 'pommes': 430, 'bananes': 312, 'poires': 137}
del (invent['pommes'])
print (invent)
# affiche {'oranges': 274, 'bananes': 312, 'poires': 137} I
```

- **len()**

La fonction len() est utilisable avec un dictionnaire : elle en renvoie le nombre d'éléments.

2.2 Parcours d'un dictionnaire

On peut utiliser une boucle **for** pour parcourir un dictionnaire, mais au cours de l'itération, ce sont les clés utilisées dans le dictionnaire qui seront successivement affectées à la variable de travail, et non les valeurs. L'ordre dans lequel les éléments seront extraits est imprévisible.

Exemple :

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

2.3 méthodes sur les dictionnaires

méthode	description	exemple
keys	renvoie la liste des clés utilisées dans le dictionnaire	print (dico.keys()) # affiche dict_keys['computer', 'keyboard', 'mouse']
values	renvoie la liste des valeurs mémorisées dans le dictionnaire	print (dico.values()) #affiche dict_values ['ordinateur', 'clavier', 'souris']
items	extrait du dictionnaire une liste équivalente de tuples	print (invent.items()) #affiche[('oranges', 27), ('bananes', 312), ('poires',137)] for key, value in invent.items(): print("Clé :",key," | valeur :", value)
Update([dict])	Mets à jour le dictionnaire. Il est possible de concaténer deux dictionnaires avec cette méthode. Les doublons sont automatiquement fusionnés.	d={'apples':1,'oranges':3,'pears': 2} ud = {'pears': 4,'grapes': 5, 'lemons': 6} d.update(ud) print(d) # {'grapes': 5, 'pears': 4, 'lemons': 6, 'apples': 1, 'oranges': 3}
clear()	Supprime tous les éléments du dictionnaire.	

2.3 méthodes sur les dictionnaires

méthode	description	exemple
copy	La méthode copy() permet d'effectuer une vraie copie d'un dictionnaire. Il faut savoir en effet que la simple affectation d'un dictionnaire existant à une nouvelle variable crée seulement une nouvelle référence vers le même objet, et non un nouvel objet (aliasing).	invent = {'bananes':312,'oranges':274,'poires':137} stock = invent print (stock) del(invent['bananes']) print (stock) #affiche{'oranges':27,'poires': 137} #Si on modifie "invent", alors stock aussi est modifié, #et vice-versa car les deux référencent le même objet **Avec copy** magasin = stock.copy() magasin['prunes'] = 561 print (magasin) #{'oranges':27,'prunes':561,'poires': 137} print(stock) #{'oranges': 274, 'poires': 137} print (invent) # {'oranges': 274, 'poires': 137}

méthodera	description	exemple

pop(clé[, d])	Supprimez l'élément avec "clé" et renvoyez sa valeur ou "d" si "clé" est introuvable. Si "d" n'est pas fourni et que "key" n'est pas trouvé, KeyError est levée.	D={"prenom":"lamia", "ville":"rades", "age":20}
		print(D) #{'prenom': 'lamia', 'ville': 'rades', 'age': 20}
		
		print("val : ", D.pop('nom','inexistant')) # val : inexistant
		print("pop : ",D)
		# pop : {'prenom': 'lamia', 'ville': 'rades', 'age': 20}

popitem()	Supprimer et renvoyer un élément arbitraire (clé, valeur). KeyError si le dictionnaire est vide.	
get(clé[, d])	Renvoie la valeur de la clé. Si la clé n'existe pas, retourne d (la valeur par défaut est None).	D={"prenom":"lamia", "ville":"rades", "age":20} print(D.get("ville")) #rades

3. Les ensembles

- Un ensemble est une collection itérable non ordonnée d'éléments hachables uniques.
- C'est la transposition informatique de la notion d'ensemble mathématique.

En python, on trouve deux types d'ensembles:

- Le **set** qui est un objet modifiable
- le **frozenset** qui est un objet immuable (le mot anglais frozen signifie gelé en français)

3.1 Déclarer un set

Il y a deux manières de déclarer un set.

- ***En utilisant des accolades***.

```
s = { "anas", "lamia", "amine", "aziz"}
print(s)        # affiche {'lamia', 'aziz', 'anas', 'amine'}
```

Notez bien qu'il s'agit d'un ensemble non ordonné ce qui explique que print() retourne les éléments dans un ordre aléatoire.

3. Les ensembles

- Un ensemble est une collection itérable non ordonnée d'éléments hashables uniques.
- C'est la transposition informatique de la notion d'ensemble mathématique.

En python, on trouve deux types d'ensembles:

- Le **set** qui est un objet modifiable
- le **frozenset** qui est un objet immuable (le mot anglais frozen signifie gelé en français)

3.1 Déclarer un set

Il y a deux manières de déclarer un set.

- ***En utilisant des accolades***

```
s = { "anas", "lamia", "amine", "aziz"}
print(s)        # affiche {'lamia', 'aziz', 'anas', 'amine'}
```

Notez bien qu'il s'agit d'un ensemble non ordonné ce qui explique que print() retourne les éléments dans **un ordre aléatoire**.

- ***En transformant une liste en set grâce au constructeur set()***

```
s2 = set(["lamia", "lamia", (1,2,3), 7.56])
 print(s2)      # affiche {7.56, 'lamia', (1, 2, 3)}
```

Notez qu'on a déclaré dans le set avec un doublon ('lamia') mais, celui-ci n'est pas pris en compte et ne lève même pas d'exception. Au fait, le e set est une collection d'objets uniques.

- Il n'est pas possible de déclarer un ensemble vide avec la première méthode. Cela créé un dictionnaire.

```
s = {}
print(type(s)) # affiche <class 'dict'>
```

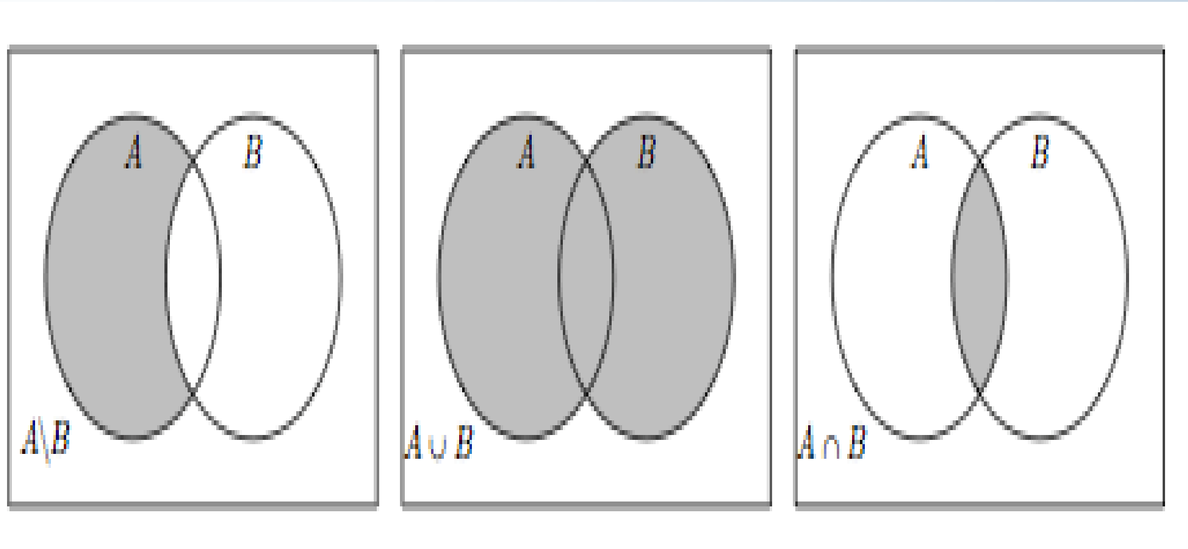
Pour créer un ensemble vide, pas d'autre choix que d'utiliser la deuxième méthode.

```
s = set()
print(type(s))  #affiche <class 'set'>
```

3.2 Aperçu de quelques opérations mathématiques associées aux sets

- ***Différence entre deux sets***

```
s = {"amine", "aziz", "anas", "ahmed"}
s2 = {"amine", "anas"}
s3 = s - s2
print("s3 =", s3)   # s3={'aziz', 'ahmed'}
```



- ***Union de deux sets***

```
s = {"Python", "php"}
s2 = {"java", "html"}
s3 = s | s2
print("s3 =", s3)# s3 = {'html', 'php', 'java', 'Python'}
```

- ***Intersection de deux sets***

```
s = {"Python", "php"}
s2 = {"java", "php"}
s3 = s & s2
print("s3 =", s3)   # s3= {'php'}
```

3.2 Quelques méthodes associées aux sets

- ***Ajouter un élément avec set.add()***

```
s = {"amine", "anas", "aziz"}
s.add("lamia")
print(s)       #affiche {‘anas’,’aziz’,’amine’,’lamia’}
```

- ***Ajouter tous les éléments d’un autre set avec set.update()***

```
s = {"amine", "ahmed", "Mohamed"}
S2 = {1, 2, 3}
s.update(s2)
print("s =", s)            # s = {1, 2, 3, ‘amine’, ‘ahmed’, ‘Mohamed’}
print("s2 =", s2)        # s2 = {1, 2, 3}
```

Comme vous pouvez le constater, cette méthode rajoute les éléments de l’ensemble s2 dans l’ensemble s sans pour autant vider l’ensemble s2.

Supprimer un élément avec set.remove() et set.discard()

```
s = {"amine", "ahmed", "Mohamed", "aziz"}
s.remove("ahmed")
print(s)          # {"amine", "ahmed", "Mohamed", "aziz"}
```

Attention! Avec set.remove(), Python lève une exception si vous essayez d'enlever un élément qui n'appartient pas au set.

```
s = {"amine", "ahmed", "Mohamed", "aziz"}
s.remove("salah")
print(s)          # on aura KeyError: 'salah'
```

Pour éviter cela, utiliser la méthode **set.discard()** qui, elle, ne lèvera pas d'exception.

```
s = {"amine", "ahmed", "Mohamed", "aziz"}
s.discard("salah")
print(s) # {"amine", "aziz", "Mohamed", "ahmed"}
```

Vider un set avec la méthode set.clear()

```
s = {"amine", "ahmed", "Mohamed", "aziz"}
s.clear()
print("s =", s)#  affiche s=set()
```

4. Les frozensets

Problèmes avec les ensembles

- Les ensembles étant modifiables sont unhashable ➔ ils ne peuvent pas être utilisés comme des clés de dictionnaires.
- On ne peut pas mettre un set dans un autre.

- Frozenset est une nouvelle classe qui présente les caractéristiques d'un ensemble, mais ses éléments ne peuvent pas être modifiés une fois affectés.➔ les méthodes telles que **add()** ou **remove()** ne peuvent pas leur être applicables.

- Les tuples sont des listes immuables, les frozensets sont des ensembles immuables**.**

- Notez bien que frozenset() fonctionne également avec un tuple ou une liste

Déclaration d'un frozenset:

Les Frozensets peuvent être créés en utilisant la fonction **frozenset().**

Exemples

```
villes = set (["tunis", "Béja", "Nabeul"])
villes.add ("Sousse")
print(villes) # affiche  {'Sousse', 'Béja', 'tunis', 'Nabeul'}
```

```
villes = frozenset (["tunis", "Béja", "Nabeul"])
villes.add ("Sousse")
print(villes)
AttributeError: 'frozenset' object has no attribute 'add'
```

```
tup = (1, 2, 3)
T_fr_s = frozenset(tup)
print(type(T_fr_s))        #<class 'frozenset’>


#une fois déclaré, un frozenset peut appartenir à un set car il est devenu immuable
liste = [1, 2, 3]
fr_s = frozenset(liste)
print(type(fr_s))        # <class ‘frozenset’>
s = {fr_s, 4, 5}
print("s =", s)         #s = {5, frozenset({1, 2, 3}), 4}
```