Université de Gabés
Faculté des Sciences de Gabés

LGLSI 1

Semestre 2

Programmation Python

Dhikra KCHAOU
dhikrafsegs@gmail.com

2023-2024

Chapitre 5:

Les ensembles et les dictionnaires

Les ensembles

Définition d'un ensemble

- Un ensemble est une collection d'objets de **différents types**, **sans répétition** **(sans doublons)** et **sans ordre** **(sans Indiçage)**.
- Un ensemble est déclaré par une série de valeurs, séparée par des virgules, et encadrée par des accolades **{ }**

Différents types + sans ordre

```
>>> ens={1,12.4,3,"AA"}
>>> ens
{'AA', 3, 12.4, 1}
```

Sans doublons

```
>>> ens={1,12.4,3,"AA",1}
>>> len(ens)
4
>>> ens
{'AA', 3, 12.4, 1}
```

Les ensembles

- Création d'un **ensemble vide** :

```
>>> ens=set()
>>> ens
set()
>>> type(ens)
<class 'set'>
```

```
>>> ens={}
>>> ens
{}
>>> type(ens)
<class 'dict'>
```

➡ Un ensemble vide se définit par **set()** et non par { }

Concaténation des ensembles

```
>>> ensl={1,4}
>>> ensl=ensl+{1,6}
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    ensl=ensl+{1,6}
TypeError: unsupported operand type(s) for +: 'set' and 'set'
```

- Concaténation (union) **par | et non plus par +**

```
>>> ensl={1,4}
>>> ensl=ensl|{1,6}
>>> ensl
{1, 4, 6}
```

```
>>> ensl={1,4}
>>> id(ensl)
2319449335264
>>> ensl=ensl|{1,6}
>>> id(ensl)
2319454442144
```

Caractère mutable des ensembles

- Un **ensemble est mutable, par contre ses éléments sont immutables**.

```
>>> ens={1,12.4,3,"AA"}
>>> ens.append(6)
Traceback (most recent call last):
  File "<pyshell#31>", line 1, in <module>
    ens.append(6)
AttributeError: 'set' object has no attribute 'append'
```

- **Il n'y a pas de méthode append() pour les ensembles.**

➡ Un SET organise ses éléments à sa manière, il n'y a pas la notion d'ordre (**append()** ajoute un élément à la fin d'une séquence)

➡ La méthode **add()** est utilisée dans le cas des ensembles.

Caractère mutable des ensembles

- Un ensemble est **mutable**:

```
>>> ens={1,12.4,3,"AA"}
>>> id(ens)
2319454442592
>>> ens.add(6)
>>> ens
{1, 3, 6, 'AA', 12.4}
>>> id(ens)
2319454442592
```

- Les éléments d'un ensemble sont **immutables** :

```
>>> ens[1]=4
Traceback (most recent call last):
  File "<pyshell#41>", line 1, in <module>
    ens[1]=4
TypeError: 'set' object does not support item assignment
```

- On ne peut pas accéder à un élément de l'ensemble en utilisant les indices :

```
>>> ens={1, 3, 6, 'AA', 12.4}
>>> ens[1]
Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    ens[1]
TypeError: 'set' object is not subscriptable
```

Ensemble de tuples

- On peut avoir **un ensemble de tuples** :

```
>>> ens={12,(55,"BC")}
>>> ens
{12, (55, 'BC')}
```

- Mais, **on ne peut pas avoir des ensembles de listes, ou d'ensembles**:

```
>>> ens={12,[55,"BC"]}
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    ens={12,[55,"BC"]}
TypeError: unhashable type: 'list'
```

```
>>> ens={12,{55,"BC"}}
Traceback (most recent call last):
  File "<pyshell#7>", line 1, in <module>
    ens={12,{55,"BC"}}
TypeError: unhashable type: 'set'
```

➡ Un ensemble contient des **objets uniques** qui sont **hashable** : **la valeur de hachage ne change pas pendant la durée de vie.**

Conversion liste, tuple, chaine vers ensemble

```
>>> ens=set([1,"aa",38,"k"])
>>> ens
{1, 'aa', 'k', 38}
>>> type(ens)
<class 'set'>
```

```
>>> ens=set([1,"aa",38,"k",1])
>>> ens
{1, 'aa', 'k', 38}
>>> type(ens)
<class 'set'>
```

```
>>> ens=set((1,"aa",38,"k",1))
>>> ens
{1, 'aa', 'k', 38}
>>> type(ens)
<class 'set'>
```

```
>>> ens= set("programme")
>>> ens
{'a', 'r', 'e', 'm', 'p', 'o', 'g'}
```

- La conversion se fait en utilisant la méthode **set()**

Les ensembles

- Génération d'un ensemble d'entiers avec **range()**:

```
>>> ens= set(range(5))
>>> ens
{0, 1, 2, 3, 4}
```

- **Ensemble par compréhension:**

```
>>> ens={i for i in range(5) if i%2==0}
>>> ens
{0, 2, 4}
```

- **Exemple d'une liste par compréhension**

```
>>> [(x, y) for x in [1,2,3] for y in [3,1,4] if x != y]
[(1, 3), (1, 4), (2, 3), (2, 1), (2, 4), (3, 1), (3, 4)]
```

Opérations sur les ensembles

- len(ens) : renvoie la longueur de l'ensemble ens.
- max(ens), min(ens), sum(ens) revoient le maximum, le minimum et la somme des éléments d'un ensemble.
- ens.add(elem) ajoute l'élément elem à l'ensemble ens.
- ens.update(elem1,elem2,…) ajoute plusieurs éléments **iterable** à l'ensemble ens.
- ens.remove(elem) supprime l'élément elem de l'ensemble ens (s'il existe).
- ens.pop() renvoie un élément aléatoire de l'ensemble ens, et le supprime.

Exemples

```
>>> e={10,12,4}
>>> len(e)
3
>>> e={10,12,4,4}
>>> len(e)
3
>>> print(max(e),min(e),sum(e),sep="--")
12--4--26
```

```
>>> e.add(13)
>>> e
{10, 13, 4, 12}
>>> e.update("G",["Z",9])
>>> e
{4, 9, 10, 'Z', 12, 13, 'G'}
```

```
>>> e.add(13,"DD")
Traceback (most recent call last):
  File "<pyshell#16>", line 1, in <module>
    e.add(13,"DD")
TypeError: set.add() takes exactly one argument (2 given)
>>> e.update(15,[3,8])
Traceback (most recent call last):
  File "<pyshell#17>", line 1, in <module>
    e.update(15,[3,8])
TypeError: 'int' object is not iterable
```

```
>>> ens={1,2}
>>> ens.update(4,7)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    ens.update(4,7)
TypeError: 'int' object is not iterable
>>> ens1={4,7}
>>> ens.update(ens1)
>>> ens
{1, 2, 4, 7}
```

Exemples

```
>>> e
{4, 9, 10, 'Z', 12, 13, 'G'}
>>> e.remove(4)
>>> e
{9, 10, 'Z', 12, 13, 'G'}
>>> e.remove(2)
Traceback (most recent call last):
  File "<pyshell#20>", line 1, in <module>
    e.remove(2)
KeyError: 2
```

```
>>> e
{9, 10, 'Z', 12, 13, 'G'}
>>> e.pop()
9
>>> e.pop()
10
>>> e
{'Z', 12, 13, 'G'}
>>> e.pop()
'Z'
```

Opérations sur les ensembles

- elem in ens, elem not in ens : vérifie l'appartenance d'un élément elem dans l'ensemble ens.
- ens.copy() copie les éléments de l'ensemble ens
- ens.clear() efface le contenu de l'ensemble ens. Elle retourne un set().
- ens1 <= ens2 ou bien ens1.issubset(ens2) ou bien ens2.issuperset(ens1) vérifie si ens1 est un sous-ensemble de ens2.
- **NB. Un ensemble ens1 est un sous-ensemble de ens2 si chaque élément de ens1 est également un élément de ens2.**

Exemples

```
>>> e={'Z',9,'G',10,13}
>>> 13 in e
    True
>>> 12 not in e
    True
>>> el=e.copy()
>>> el
    {'Z', 'G', 9, 10, 13}
>>> e.clear()
>>> e
    set()
>>> set(['Z',10]) in el
```

```
>>> ens1={10,7}
>>> ens2={1,10,7,22}
>>> ens1.issubset(ens2)
    True
```

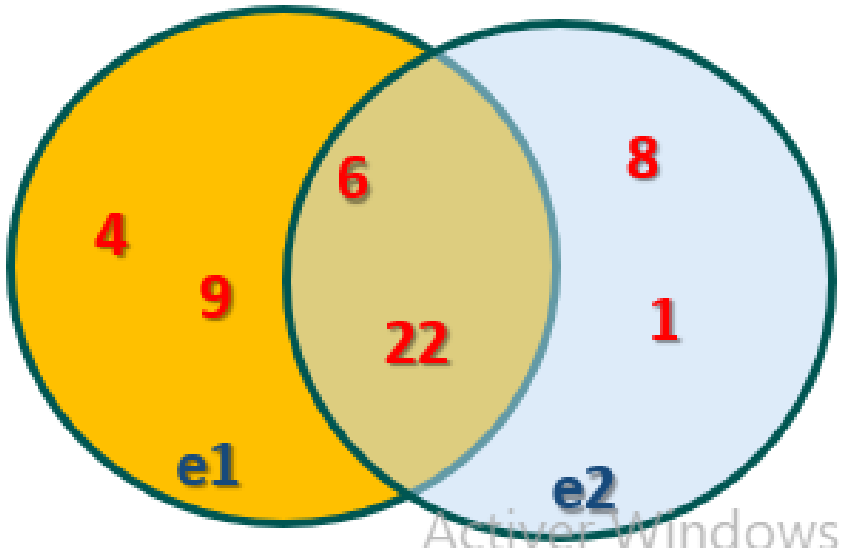
```
>>> ens={1,7,8}
>>> ens1={1,7}
>>> ens1 in ens
    False
>>> ens1.issubset(ens)
    True
```

Opérations sur les ensembles

- ens1==ens2 retourne True si les ensembles ens1 et ens2 contiennent les mêmes éléments.
- ens1!=ens2 retourne True si les ensembles ens1 et ens2 contiennent au moins un élément différent.
- ens1|ens2 ou bien ens1.union(ens2) retourne un ensemble qui est l'union des ensembles ens1 et ens2.
- ens1&ens2 ou bien ens1.intersection(ens2) retourne un ensemble qui est l'intersection des ensembles ens1 et ens2.
- ens1-ens2 ou bien ens1.difference(ens2) retourne un ensemble qui représente les éléments qui existe dans ens1 et n'existe pas ens2.
- ens1^ens2 ou bien ens1.symmetric_difference(ens2) retourne un ensemble qui représente les éléments qui existe dans ens1 et n'existe pas ens2 et les éléments qui existe dans ens2 et n'existe pas dans ens1.

Exemples

```
>>> e1=set([4,6,9,22])
>>> e2=set([1,6,8,22])
>>> e1==e2
False
>>> e1|e2
{1, 4, 6, 8, 9, 22}
>>> e1&e2
{6, 22}
>>> e1-e2
{9, 4}
>>> e2-e1
{8, 1}
>>> e1^e2
{1, 4, 8, 9}
```



Parcourir un ensemble

- **Remarques** :
  - Pour vérifier si un élément existe ou non dans un ensemble, on utilise l'opérateur **in : 10 in ens → true**
  - On peut utiliser **enumerate()** pour parcourir un ensemble en python, cependant, l'index généré ne représente pas l'index de l'élément affiché mais l'ordre dans lequel les éléments sont parcourus.

```
>>> ens={10,2,'Z',40}
>>> for value in ens:
...     print(value)
...
...
...
10
40
Z
2
```

```
>>> my_set = {1, 2, 3, 4, 5}
>>> for index, value in enumerate(my_set):
...     print(index, value)
...
...
0 1
1 2
2 3
3 4
4 5
```

Exercice

- Écrire un programme en python qui prend en entré un texte T donné et génère un ensemble A formé des mots contenant la lettre 'a' et l'ensemble B formé des mots ne contenant pas la lettre 'a'.
- L'ensemble B doit être généré par deux façons différentes:

1. En faisant une manipulation directe sur le texte T
2. En faisant le passage au complémentaire de A.

- Exemple T= «Python est un langage de programmation universel »

A : {'programmation', 'langage'}

B: {'est', 'de', 'python', 'un', 'universel'}

Correction

- **1ére méthode** :

```
T="python est un langage de programmation universel"
words=T.split()
A={word for word in words if 'a' in word}
B={word for word in words if 'a' not in word}
print(A)
print(B)
```

- **2ème méthode**:

```
T="python est un langage de programmation universel"
words=T.split()
A={word for word in words if 'a' in word}
#B=set(words).difference(A)
B=set(words)-A
print(A)
print(B)
```

Les dictionnaires

Les dictionnaires

- Les dictionnaires représentent une structure de type **clé:valeur.**
  - Les clés doivent être de type **immuable**,
  - Les valeurs peuvent être de n'importe quel type, y compris des objets mutables.
- Les éléments d'un dictionnaire sont **non ordonnés**.
- Exemple:

```
>>> d={"Nom":"Sami","age":12}
>>> d
{'Nom': 'Sami', 'age': 12}
```

Les dictionnaires

- Création d'un dictionnaire vide :

```
>>> d=dict()
>>> d
{}
>>> type(d)
<class 'dict'>
```

Ou bien

```
>>> d={}
>>> d
{}
>>> type(d)
<class 'dict'>
```

- Le seul moyen d'accès à une valeur particulière est par l'intermédiaire de sa clé: d[cle] renvoie la valeur associée à la clé cle du dictionnaire d.

```
>>> dl={'Nom':'Ali','Age':12,'Notes':[10.5,13]}
>>> dl["Nom"]
'Ali'
>>> dl["Notes"]
[10.5, 13]
>>> dl['Adresse']
Traceback (most recent call last):
  File "<pyshell#14>", line 1, in <module>
    dl['Adresse']
KeyError: 'Adresse'
```

Opérations sur les dictionnaires

- **Ajout et mise à jour** d'une valeur dans un dictionnaire :
  - dic[cle]=elem modifie la valeur associée à la clé cle à elem (si cle existe) **sinon** ajoute cle:elem au dictionnaire dic

```
>>> dl=dict()
>>> dl['id']=103
>>> dl
{'id': 103}
>>> dl['Nom']="Slim"
>>> dl
{'id': 103, 'Nom': 'Slim'}
>>> dl["Nom"]="Mohamed"
>>> dl
{'id': 103, 'Nom': 'Mohamed'}
```

← Ajout d'un élément de clé ''Nom'' et de valeur ''Slim''

← MAJ la valeur de l'élément de clé « Nom » à ''Mohamed"

  - dic.update({cle:val}) ajoute un élément {cle:val} au dictionnaire dic si la clé n'existe pas. La fonction update() fait la MAJ si la clé existe.

Exemple sur la fonction update()

```
>>> dl=dict()
>>> dl['id']=103
>>> dl
{'id': 103}
>>> dl["Nom"]="Slim"
>>> dl
{'id': 103, 'Nom': 'Slim'}
>>> dl.update({'Salaire':1000})
>>> dl
{'id': 103, 'Nom': 'Slim', 'Salaire': 1000}
>>> dl.update({'Age':20})
>>> dl
{'id': 103, 'Nom': 'Slim', 'Salaire': 1000, 'Age': 20}
>>> dl.update({'Age':21})
>>> dl
{'id': 103, 'Nom': 'Slim', 'Salaire': 1000, 'Age': 21}
```

Opérations sur les dictionnaires

- len(dic) renvoie la longueur du dictionnaire dic (nombre de clés).
- cle in dic renvoie True si la clé cle est présente dans le dictionnaire dic, False sinon.
- dic.copy() renvoie une copie du dictionnaire dic, indépendante de l'original.
- dic.pop(cle) renvoie la valeur et supprime le paire **cle:valeur**.
- del dic[cle] efface un paire **cle:valeur**.
- dic.clear() efface le contenu du dictionnaire dic.

Exemples

```
>>> dict={"chien":"dog","souris":"mouse"}
>>> len(dict)
2
>>> "chien" in dict
True
>>> "Chien" in dict
False
>>> "dog" in dict
False
```

```
>>> dict={"chien":"dog","souris":"mouse"}
>>> dc=dict.copy()
>>> dc
{'chien': 'dog', 'souris': 'mouse'}
>>> id(dict)
2197097885824
>>> id(dc)
2197138021056
```

Exemples

```
>>> dict={"chien":"dog","souris":"mouse"}
>>> dict.pop()
Traceback (most recent call last):
  File "<pyshell#6>", line 1, in <module>
    dict.pop()
TypeError: pop expected at least 1 argument, got 0
>>> dict.pop("chien")
'dog'
>>> dict
{'souris': 'mouse'}
```

```
>>> dictl={"chien":"dog","souris":"mouse"}
>>> del dictl ["chien"]
>>> dictl
{'souris': 'mouse'}
```

```
>>> dictl
{'souris': 'mouse'}
>>> dictl.clear()
>>> dictl
{}
```

Parcours d'un dictionnaire

- **dic.items()** renvoie un **itérable** pour décrire les couples cle:valeur du dictionnaire dic dans une boucle for

```
>>> dict={"chien":"dog","souris":"mouse"}
>>> for i in dict.items():
...     print(i)
...
...
('chien', 'dog')
('souris', 'mouse')
```

- **dic.keys()** renvoie un **itérable** pour décrire **les clés** de dic dans une boucle for

```
>>> for i in dict.keys():
...     print(i,end="-")
...
...
chien-souris-
```

- **dic.values()** renvoie un **itérable** pour décrire **les valeurs** de dic dans une boucle for

```
>>> for i in dict.values():
...     print(i,end="-")
...
...
dog-mouse-
```

Existence d'une clé

- Pour vérifier si l'une des clés existe dans un dictionnaire, on peut utiliser le test d'appartenance avec l'instruction **in** qui renvoie un booléen:

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> if "Ali" in Moyennes:
...     print("La clé 'Ali' existe dans le dict")
...
...
La clé 'Ali' existe dans le dict
```

Existence d'une valeur

- Pour vérifier si l'une des clés a une valeur donnée dans un dictionnaire, on peut utiliser le test d'appartenance avec l'instruction in qui renvoie un booléen et la méthode **values()** qui retourne les valeurs dans un dictionnaire:

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> if 17 in Moyennes.values():
...     print("La valeur 17 existe dans cette classe")
...
...
La valeur 17 existe dans cette classe
```

Méthode .get()

- La méthode **.get()** extrait la valeur associée à une clé mais ne renvoie pas d'erreur si la clé n'existe pas :

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> Moyennes.get("Ali")
8
>>> Moyennes.get("Salah")
>>>
```

- Cas de clé non existante:

```
>>> print(Moyennes.get("Salah"))
None
>>> if(Moyennes.get("Salah")==None):
...     print("Non existant")
...
...
Non existant
```

- On peut également indiquer à **.get()** une valeur par défaut si la clé n'existe pas :

```
>>> print(Moyennes.get("Salah", "Non existant"))
Non existant
```

Tri d'un dictionnaire par clés

- Pour trier un dictionnaire **par ses clés** on peut utiliser la fonction **sorted()**:

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> sorted(Moyennes)
['Ali', 'Asma', 'Mohamed']
```

- Les **clés** sont triées ici par ordre alphabétique croissant.

Tri d'un dictionnaire par valeur

- Pour trier un dictionnaire par ses valeurs, il faut utiliser la fonction **sorted()** avec l'argument **key**:

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> sorted(Moyennes, key=Moyennes.get)
['Ali', 'Mohamed', 'Asma']
```

- **Attention, ce sont les clés du dictionnaires qui sont renvoyées, pas les valeurs.**

Ces clés sont renvoyées dans un ordre qui permet d'obtenir les clés triées par ordre croissant des valeurs associées:

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> for key in sorted(Moyennes, key=Moyennes.get):
...     print(key,Moyennes[key])
...
...
Ali 8
Mohamed 12
Asma 17
```

Clé associé au maximum ou au minimum des valeurs

- Les fonctions **min()** et **max()** acceptent également l'argument **key=**. On peut ainsi obtenir la clé associée au minimum ou au maximum des valeurs d'un dictionnaire :

```
>>> Moyennes={ 'Ali' :8,'Mohamed' :12,'Asma' :17}
>>> max(Moyennes, key=Moyennes.get)
'Asma'
>>> min(Moyennes, key=Moyennes.get)
'Ali'
```

Liste de dictionnaires

- En créant une liste de dictionnaires qui possèdent les mêmes clés, on obtient une structure qui ressemble à une base de données:

```
>>> etud1={"Nom":"Mohamed","Age":20,"Moyenne":15}
>>> etud2={"Nom":"Rahma","Age":19,"Moyenne":12}
>>> etudiants=[etud1,etud2]
>>> etudiants
[{'Nom': 'Mohamed', 'Age': 20, 'Moyenne': 15}, {'Nom': 'Rahma', 'Age': 19,
'Moyenne': 12}]
>>> for etud in etudiants:
...     print(etud["Nom"])
...
...
Mohamed
Rahma
>>>
```

Exercice 1

1. Soit $\boldsymbol{d}$ un dictionnaire. Si la clé k n'est pas dans le dictionnaire, l'instruction $d[k]$=5 provoque-t-elle une erreur ?
2. Soit $\boldsymbol{d}$ un dictionnaire. L'instruction $d[k]$=[10,5] provoque-t-elle une erreur ?
3. Soit $\boldsymbol{d}$ un dictionnaire. L'instruction $d[[10,5]]$=5 provoque-t-elle une erreur ?