Informatique S3 : Dictionnaires Ensembles

L2 MIASHS

Laurent Boyer

2024

Section 1

Ensembles

Ensembles

- **Implémente la notion mathématique d'ensemble.**

= une structure de donnée non ordonnée, et sans doublon.

Ensembles

- **Implémente la notion mathématique d'ensemble.**
= une structure de donnée non ordonnée, et sans doublon.

- **Ecriture littérale :** des éléments séparés par des virgules, entre accolades.

```
1 x = {1, 3, 2, 3, 1}
2 print(x)
3 print(len(x))
```

▼

Ensembles

▶ **Implémente la notion mathématique d'ensemble.**
= une structure de donnée non ordonnée, et sans doublon.

▶ **Ecriture littérale :** des éléments séparés par des virgules, entre accolades.

```
1 x = {1, 3, 2, 3, 1}
2 print(x)
3 print(len(x))
```

▼

```
{1, 2, 3}
3
```

Ensembles

▶ **Implémente la notion mathématique d'ensemble.**
= une structure de donnée non ordonnée, et sans doublon.

▶ **Ecriture littérale :** des éléments séparés par des virgules, entre accolades.

```
1 x = {1, 3, 2, 3, 1}
2 print(x)
3 print(len(x))
```

▼

```
{1, 2, 3}
3
```

▶ `len` compte le nombre d'éléments.

Cas concret : Détection de doublons dans une base de données

- **Problème :** Vous avez une liste d'utilisateurs et souhaitez supprimer les doublons.

Cas concret : Détection de doublons dans une base de données

▶ **Problème :** Vous avez une liste d'utilisateurs et souhaitez supprimer les doublons.

```
# Détection de doublons dans une base de données d'utilisat
utilisateurs = ["Alice", "Bob", "Alice", "Charles"]
utilisateurs_uniques = set(utilisateurs) # Conversion en en
print(utilisateurs_uniques)
```

Cas concret : Détection de doublons dans une base de données

- **Problème :** Vous avez une liste d'utilisateurs et souhaitez supprimer les doublons.

```
# Détection de doublons dans une base de données d'utilisat
utilisateurs = ["Alice", "Bob", "Alice", "Charles"]
utilisateurs_uniques = set(utilisateurs) # Conversion en en
print(utilisateurs_uniques)
```

- **Utilité :** Les ensembles en Python sont parfaits pour garantir l'unicité des éléments.

Cas concret : Détection de doublons dans une base de données

- **Problème :** Vous avez une liste d'utilisateurs et souhaitez supprimer les doublons.

```
# Détection de doublons dans une base de données d'utilisat
utilisateurs = ["Alice", "Bob", "Alice", "Charles"]
utilisateurs_uniques = set(utilisateurs) # Conversion en en
print(utilisateurs_uniques)
```

- **Utilité :** Les ensembles en Python sont parfaits pour garantir l'unicité des éléments.

```
{'Alice', 'Charles', 'Bob'}
```

Cas concret : Comparaison d'événements

- **Problème :** Comparer les participants à deux événements pour trouver les personnes communes.

Cas concret : Comparaison d'événements

▶ **Problème :** Comparer les participants à deux événements pour trouver les personnes communes.

```
# Comparaison des participants à deux événements
event1 = {"Alice", "Bob", "David"}
event2 = {"Charles", "Bob", "Emily"}
communs = event1.intersection(event2) # Intersection pour t
print(communs)
```

Cas concret : Comparaison d'événements

- **Problème :** Comparer les participants à deux événements pour trouver les personnes communes.

```
1 # Comparaison des participants à deux événements
2 event1 = {"Alice", "Bob", "David"}
3 event2 = {"Charles", "Bob", "Emily"}
4 communs = event1.intersection(event2) # Intersection pour t
5 print(communs)
```

- **Utilité :** Les ensembles facilitent la comparaison rapide grâce aux opérations d'union, intersection et différence.

Cas concret : Comparaison d'événements

▶ **Problème :** Comparer les participants à deux événements pour trouver les personnes communes.

```
# Comparaison des participants à deux événements
event1 = {"Alice", "Bob", "David"}
event2 = {"Charles", "Bob", "Emily"}
communs = event1.intersection(event2) # Intersection pour t
print(communs)
```

- ▶ **Utilité :** Les ensembles facilitent la comparaison rapide grâce aux opérations d'union, intersection et différence.

```
{'Bob'}
```

Opérations ensemblistes (1)

```
1 set1 = {1, 3, 5}
2 set2 = {3, 1, 1, 5}
3 set3 = {1, 2, 3, 3, 1}
4
5 print(set1 == set2) # test de l'égalité
6 print(set1 & set3 , set1 | set3) # intersection et union
7 print(set1 < set3) # test de l'inclusion
8 print(3 in set1) # test de l'appartenance, mot-clé in
```

Opérations ensemblistes (1)

```
1 set1 = {1, 3, 5}
2 set2 = {3, 1, 1, 5}
3 set3 = {1, 2, 3, 3, 1}
4
5 print(set1 == set2) # test de l'égalité
6 print(set1 & set3 , set1 | set3) # intersection et union
7 print(set1 < set3) # test de l'inclusion
8 print(3 in set1) # test de l'appartenance, mot-clé in
```

▼

```
True
{1, 3} {1, 2, 3, 5}
False
True
```

Opérations ensemblistes (2)

```
1 set1 = {1, 3, 5}
2 set2 = {3, 1, 1, 5}
3
4 print(set1)
5 set1.add(6) # ajout ensembliste
6 set1.discard(12) # retrait (sans erreur)
7 set1.remove(1) # retrait (avec erreur si absent)
8 print(set1)
```

Opérations ensemblistes (2)

```
1 set1 = {1, 3, 5}
2 set2 = {3, 1, 1, 5}
3
4 print(set1)
5 set1.add(6) # ajout ensembliste
6 set1.discard(12) # retrait (sans erreur)
7 set1.remove(1) # retrait (avec erreur si absent)
8 print(set1)
```

```
{1, 3, 5}
{3, 5, 6}
```

Autres opérations

- **Autres méthodes de création**, avec `set`
  - ensemble vide
  - à partir de liste ou autre
- **Parcours** `for x in e`

```
1 x = set() # ensemble vide
2 y = set([1,3,2,2,3,3]) # ensemble des éléments de la liste
3 x.update([1,2]) # ajout d'un ensemble d'éléments
4 for a in y : # parcours pour chaque a de y
5    print(a)
6 print(x)
```

▼

Autres opérations

- **Autres méthodes de création**, avec `set`
  - ensemble vide
  - à partir de liste ou autre
- **Parcours** `for x in e`

```
1 x = set() # ensemble vide
2 y = set([1,3,2,2,3,3]) # ensemble des éléments de la liste
3 x.update([1,2]) # ajout d'un ensemble d'éléments
4 for a in y : # parcours pour chaque a de y
5    print(a)
6 print(x)
```

▼

```
1
2
3
{1, 2}
```

Exemple 1

- Comment déterminer que deux listes ont les mêmes éléments ?

Exemple 1

▶ Comment déterminer que deux listes ont les mêmes éléments ?

```
1 def memeEnsemble(l1,l2) :
2     return set(l1)==set(l2)
```

Exemple 1

▶ Comment déterminer que deux listes ont les mêmes éléments ?

```
1 def memeEnsemble(l1,l2) :
2     return set(l1)==set(l2)
```

Exemple 2 (1/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Exemple 2 (1/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Listes : Créer une fonction qui prend une liste `l` en paramètre et retourne une liste contenant tous les éléments de `l` (sans doublons, et en ne regardant pas l'ordre).

Exemple 2 (1/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Listes : Créer une fonction qui prend une liste `l` en paramètre et retourne une liste contenant tous les éléments de `l` (sans doublons, et en ne regardant pas l'ordre).

```
1 def listesansdoublon(l) :
2     h = []
3     for x in l :
4         if x not in h :
5             h.append(x)
6     return h
```

Exemple 2 (1/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Listes : Créer une fonction qui prend une liste `l` en paramètre et retourne une liste contenant tous les éléments de `l` (sans doublons, et en ne regardant pas l'ordre).

```
1 def listesansdoublon(l) :
2     h = []
3     for x in l :
4         if x not in h :
5             h.append(x)
6     return h
```

```
1 print(listesansdoublon([1,4,2,1,3,2]))
2 -> [1,4,2,3]
```

Exemple 2 (1/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Listes : Créer une fonction qui prend une liste `l` en paramètre et retourne une liste contenant tous les éléments de `l` (sans doublons, et en ne regardant pas l'ordre).

```
1 def listesansdoublon(l) :
2     h = []
3     for x in l :
4         if x not in h :
5             h.append(x)
6     return h
```

```
1 print(listesansdoublon([1,4,2,1,3,2]))
2 -> [1,4,2,3]
```

- remarque : le if in cache une boucle qui parcours la liste.

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

En utilisant les ensembles :

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

En utilisant les ensembles :

```
1 def listesansdoublon(l) :
2     h = set()
3     for x in l :
4         h.add(x)
5     return h
```

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

En utilisant les ensembles :

```
1 def listesansdoublon(l) :
2     h = set()
3     for x in l :
4         h.add(x)
5     return h
```

```
1 print(listesansdoublon([1,4,2,1,3,2]))
2 -> {1,4,2,3}
```

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

En utilisant les ensembles :

```
1 def listesansdoublon(l) :
2     h = set()
3     for x in l :
4         h.add(x)
5     return h
```

```
1 print(listesansdoublon([1,4,2,1,3,2]))
2 -> {1,4,2,3}
```

- remarque : c'est beaucoup plus efficace. Pourquoi ?

Exemple 2 (2/2)

- **Créer l'ensemble contenant les éléments d'une liste**

En utilisant les ensembles :

```
1 def listesansdoublon(l) :
2     h = set()
3     for x in l :
4         h.add(x)
5     return h
```

```
1 print(listesansdoublon([1,4,2,1,3,2]))
2 -> {1,4,2,3}
```

- remarque : c'est beaucoup plus efficace. Pourquoi ?
- c'est ce que fait `set`

Exemple 3

- **Sommer les éléments d'un ensemble**

Exemple 3

- **Sommer les éléments d'un ensemble**

```
1 def sommer(s) :
2     acc = 0
3     for x in s :
4         acc += x
5     return acc
```

Exemple 3

- **Sommer les éléments d'un ensemble**

```
1  def sommer(s) :
2      acc = 0
3      for x in s :
4          acc += x
5      return acc
```

  - Principale manière d'accéder aux éléments d'un ensemble = boucle `for in`.
    (pas de `h[1]`, pas de boucle sur les indices).

Exemple 3

- **Sommer les éléments d'un ensemble**

```
1  def sommer(s) :
2      acc = 0
3      for x in s :
4          acc += x
5      return acc
```

- Principale manière d'accéder aux éléments d'un ensemble = boucle `for in`.
  (pas de `h[1]`, pas de boucle sur les indices).
- Peut-on utiliser cette fonction avec une liste ?

Section 2

Dictionnaires

Dictionnaires, définition

- Il s'agit d'un ensemble de couples `cle : valeur`,
  - très efficace pour retrouver la `valeur` associée à une `cle`.

Dictionnaires, définition

- Il s'agit d'un ensemble de couples `cle : valeur`,
  - très efficace pour retrouver la `valeur` associée à une `cle`.

- Manipulations de base :

```
1 x = {'truc': 3, 'bidule': 5} # definition littérale
2 print(len(x)) # nombre d'éléments
3 print(x['truc']) # accès à un élément
4 x['machin'] = 7 # création d'une nouvelle entrée
5 del x['truc'] # suppression d'une entrée
6 x['bidule'] = 3 # modification d'une valeur
7 print(x) # afficahge d'un dico
```

Dictionnaires, définition

- Il s'agit d'un ensemble de couples `cle : valeur`,
  - très efficace pour retrouver la `valeur` associée à une `cle`.

- Manipulations de base :

```
1 x = {'truc': 3, 'bidule': 5} # definition littérale
2 print(len(x)) # nombre d'éléments
3 print(x['truc']) # accès à un élément
4 x['machin'] = 7 # création d'une nouvelle entrée
5 del x['truc'] # suppression d'une entrée
6 x['bidule'] = 3 # modification d'une valeur
7 print(x) # afficahge d'un dico
```

▼

```
2
3
{'bidule': 3, 'machin': 7}
```

Types

- Les types des clés et valeurs peuvent être variés.
  - Les clés ne peuvent être que de type `hashable`.
    - entiers
    - chaines de caractères
    - ...

Types

- Les types des clés et valeurs peuvent être variés.
  - Les clés ne peuvent être que de type `hashable`.
    - entiers
    - chaines de caractères
    - ...

- Quelques exemples :

```
1 x = {'truc': 3, 'bidule': 5}
2
3 arbre = {0:[1 ,2] ,1:[3 ,4] ,2:[5 ,6] ,3:[7 ,8] ,
4 4:[9 ,10] , 5:[11 ,12] ,6:[13 ,14] ,10:[21 ,22] ,
5 11:[23 ,24]}
6
7 params = {"server":"mpilgrim", "database":"master",
8 "uid":"sa", "pwd":"secret"}
```

Cas concret : données utilisateur

- **Problème :** Stocker des informations structurées sur des utilisateurs, comme leur nom et âge.

Cas concret : données utilisateur

▶ **Problème :** Stocker des informations structurées sur des utilisateurs, comme leur nom et âge.

```
# Stockage d'informations sur des utilisateurs
utilisateurs = {
    "u123": {"nom": "Alice", "âge": 25},
    "u456": {"nom": "Bob", "âge": 30}
}
print(utilisateurs["u123"]["nom"]) # Accès aux informations
```

Cas concret : données utilisateur

▶ **Problème :** Stocker des informations structurées sur des utilisateurs, comme leur nom et âge.

```
# Stockage d'informations sur des utilisateurs
utilisateurs = {
    "u123": {"nom": "Alice", "âge": 25},
    "u456": {"nom": "Bob", "âge": 30}
}
print(utilisateurs["u123"]["nom"]) # Accès aux informations
```

- **Utilité :** Les dictionnaires permettent un accès rapide à des informations structurées par clé.

Cas concret : données utilisateur

▶ **Problème :** Stocker des informations structurées sur des utilisateurs, comme leur nom et âge.

```
# Stockage d'informations sur des utilisateurs
utilisateurs = {
    "u123": {"nom": "Alice", "âge": 25},
    "u456": {"nom": "Bob", "âge": 30}
}
print(utilisateurs["u123"]["nom"]) # Accès aux informations
```

- ▶ **Utilité :** Les dictionnaires permettent un accès rapide à des informations structurées par clé.

```
Alice
```

Cas concret : Dictionnaire des occurrences

- **Problème :** Compter le nombre d'occurrences de chaque mot dans un texte.

Cas concret : Dictionnaire des occurrences

- **Problème :** Compter le nombre d'occurrences de chaque mot dans un texte.

```
1 texte = "bonjour bonjour tout le monde"
2 compteur = {}
3 for mot in texte.split():
4     if mot in compteur:
5         compteur[mot] += 1
6     else:
7         compteur[mot] = 1
8 print(compteur)
```

Cas concret : Dictionnaire des occurrences

- **Problème :** Compter le nombre d'occurrences de chaque mot dans un texte.

```
1 texte = "bonjour bonjour tout le monde"
2 compteur = {}
3 for mot in texte.split():
4     if mot in compteur:
5         compteur[mot] += 1
6     else:
7         compteur[mot] = 1
8 print(compteur)
```

- **Utilité :** Les dictionnaires sont efficaces pour compter des éléments en associant une clé à une valeur (le mot et son nombre d'occurrences).

Cas concret : Dictionnaire des occurrences

- **Problème :** Compter le nombre d'occurrences de chaque mot dans un texte.

```
texte = "bonjour bonjour tout le monde"
compteur = {}
for mot in texte.split():
    if mot in compteur:
        compteur[mot] += 1
    else:
        compteur[mot] = 1
print(compteur)
```

- **Utilité :** Les dictionnaires sont efficaces pour compter des éléments en associant une clé à une valeur (le mot et son nombre d'occurrences).

```
{'bonjour': 2, 'tout': 1, 'le': 1, 'monde': 1}
```

Compréhension et Parcours

```
1  x = { i: i ** 2 + 1 for i in range(5)}
2
3  print(x)
4  # parcours des clés (1)
5  for k in x:
6     print('clé :', k)
```

Compréhension et Parcours

```
1 x = { i: i ** 2 + 1 for i in range(5)}
2
3 print(x)
4 # parcours des clés (1)
5 for k in x:
6     print('clé :', k)
```

▼

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
clé : 0
clé : 1
clé : 2
clé : 3
clé : 4
```

Compréhension et Parcours

```
1  x = { i: i ** 2 + 1 for i in range(5)}
2
3  print(x)
4  # parcours des clés (2)
5  for k in x.keys():
6     print('clé :', k, 'valeur :', x[k])
```

▼

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
clé : 0 valeur : 1
clé : 1 valeur : 2
clé : 2 valeur : 5
clé : 3 valeur : 10
clé : 4 valeur : 17
```

Compréhension et Parcours

```
1  x = { i: i ** 2 + 1 for i in range(5)}
2
3  print(x)
4  # parcours des clés (2)
5  for k in x.keys():
6     print('clé :', k, 'valeur :', x[k])
```

▼

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
clé : 0 valeur : 1
clé : 1 valeur : 2
clé : 2 valeur : 5
clé : 3 valeur : 10
clé : 4 valeur : 17
```

Compréhension et Parcours

```
1 x = { i: i ** 2 + 1 for i in range(5)}
2
3 print(x)
4 # parcours des valeurs
5 for val in x.values():
6    print('valeur :', val)
```

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
valeur : 1
valeur : 2
valeur : 5
valeur : 10
valeur : 17
```

Compréhension et Parcours

```
1 x = { i: i ** 2 + 1 for i in range(5)}
2
3 print(x)
4 # parcours des valeurs
5 for val in x.values():
6    print('valeur :', val)
```

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
valeur : 1
valeur : 2
valeur : 5
valeur : 10
valeur : 17
```

Compréhension et Parcours

```
1  x = { i: i ** 2 + 1 for i in range(5)}
2
3  print(x)
4  # parcours des clés ET valeurs
5  for cle, val in x.items():
6     print('clé, valeur :', cle, val)
```

▼

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
clé, valeur : 0 1
clé, valeur : 1 2
clé, valeur : 2 5
clé, valeur : 3 10
clé, valeur : 4 17
```

Compréhension et Parcours

```
1  x = { i: i ** 2 + 1 for i in range(5)}
2
3  print(x)
4  # parcours des clés ET valeurs
5  for cle, val in x.items():
6     print('clé, valeur :', cle, val)
```

▼

```
{0: 1, 1: 2, 2: 5, 3: 10, 4: 17}
clé, valeur : 0 1
clé, valeur : 1 2
clé, valeur : 2 5
clé, valeur : 3 10
clé, valeur : 4 17
```

Exemple

- Construire le dictionnaire des occurences des caractères d'une chaîne de caractère:

Exemple

▶ Construire le dictionnaire des occurences des caractères d'une chaîne de caractère:
= le dictionnaire dont les clés sont les lettres et la valeur associée à chacune son nombre d'occurence dans la chaîne.

```
1 def occu(s) :
2     d={}
3     for c in s :
4         if c not in d :
5             d[c] = 1
6         else :
7             d[c] += 1
8     return d
9 print(occu("Bonjour à tous, cette fonction compte."))
```

▼

Exemple

▶ Construire le dictionnaire des occurences des caractères d'une chaîne de caractère:
= le dictionnaire dont les clés sont les lettres et la valeur associée à chacune son nombre d'occurence dans la chaîne.

```
1 def occu(s) :
2     d={}
3     for c in s :
4         if c not in d :
5             d[c] = 1
6         else :
7             d[c] += 1
8     return d
9 print(occu("Bonjour à tous, cette fonction compte."))
```

▼

```
{'B': 1, 'o': 6, 'n': 3, 'j': 1, 'u': 2, 'r': 1, ' ': 5, 'à
```

Généralisation à partir de l'exemple

- "Construire un *truc* à partir d'un *machin* :"

ou d'informations extraites du *machin*...

Généralisation à partir de l'exemple

▶ "Construire un *truc* à partir d'un *machin* :"
ou d'informations extraites du *machin*...

Structure générale de ce type de fonctions :

Généralisation à partir de l'exemple

- "Construire un *truc* à partir d'un *machin* :"

ou d'informations extraites du *machin*...

Structure générale de ce type de fonctions :

- initialiser le *truc* (souvent vide)

  `{}, set(), []`

Généralisation à partir de l'exemple

▶ "Construire un *truc* à partir d'un *machin* :"
ou d'informations extraites du *machin*...

Structure générale de ce type de fonctions :

- initialiser le *truc* (souvent vide)
  `{}, set(), []`
- parcourir les éléments du *machin*
  `for x in l, for x in d.items(), ...`

Généralisation à partir de l'exemple

▶ "Construire un *truc* à partir d'un *machin* :"
ou d'informations extraites du *machin*...

Structure générale de ce type de fonctions :

- initialiser le *truc* (souvent vide)
  `{}, set(), []`
- parcourir les éléments du *machin*
  `for x in l, for x in d.items(), ...`
  - à chaque étape mettre à jour, si besoin, le *truc*

Généralisation à partir de l'exemple

▶ "Construire un *truc* à partir d'un *machin* :"
ou d'informations extraites du *machin*...

Structure générale de ce type de fonctions :

- initialiser le *truc* (souvent vide)
  `{}, set(), []`
- parcourir les éléments du *machin*
  `for x in l, for x in d.items(), ...`
  - à chaque étape mettre à jour, si besoin, le *truc*
- retourner le *truc*

Pour conclure

- Pour vendredi vous pouvez reprendre tout ça et apprendre les commandes de base (écrites sur les slides)