

Séance 1

Ensemble, dictionnaire et base de données

Sébastien Combéfis, Quentin Lurkin

lundi 14 septembre 2015



Ce(tte) œuvre est mise à disposition selon les termes de la Licence Creative Commons Attribution – Pas d'Utilisation Commerciale – Pas de Modification 4.0 International.

Informations générales

- IN2T Informatique appliquée, IN2L Concepts informatiques

10 cours de 1h30 (15 heures) et 8 labos de 3h30 (28 heures)

- Documents utilisés sont sur Eole (slides et énoncés des labos)

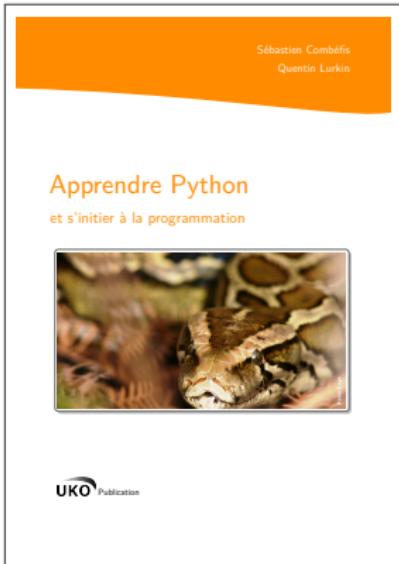
- Évaluation

- Examen (IN2T) : 50%
- Labo (IN2L) : 50% (40% évaluation continue, 10% projet)

- Enseignants

- Sébastien Combéfis (s.combefis@ecam.be)
- André Lorge (a.lorge@ecam.be)
- Quentin Lurkin (q.lurkin@ecam.be)

Livre de référence



- **Apprendre Python et s'initier à la programmation**
par *Sébastien Combéfis et Quentin Lurkin*

- Disponible en ligne :
<http://www.ukonline.be/cours/python>

Plateforme d'entraînement



- **Pythia**
Plateforme d'exercices
avec correction automatique

- Disponible en ligne (version beta) :
<http://pythia.ecam.be>

Laboratoires

- **Huit séances** de laboratoire
 - Travail sur des petits exercices de base (14/20)
 - Exercices avancés de réflexion (6/20)
- **Faire vérifier** son travail avant la fin de la séance
- **Mini projet** par binôme à travailler durant les labos

Remettre code documenté en fin de semestre

Objectifs

- Rappel des **bases de la programmation** en Python
 - Types de données, variables, affectation
 - Booléen et condition, `if-elif-else`, `while`
 - Définition et appel de fonction, récursion, sous-problèmes
 - Séquence, liste et chaînes de caractères, `for`
- **Structures de données** avancées
 - **Ensemble**, opération ensemblistes
 - **Dictionnaire**, itérations
 - Introduction aux **base de données**, format JSON

Rappels

paintbrushes?



Programme Python

■ Calcul des racines d'un trinôme du second degré

Commentaire, importation de module, donnée, variable, expression, opérateur arithmétique, fonction print

```
1 from cmath import sqrt
2
3 # Coefficients du trinôme
4 a = 1
5 b = -4
6 c = 2e2
7
8 # Calcul du discriminant
9 delta = b ** 2 - 4 * a * c
10
11 # Calcul des deux racines
12 x1 = (-b + sqrt(delta)) / (2 * a)
13 x2 = (-b - sqrt(delta)) / (2 * a)
14
15 print('Les deux racines sont :', x1, 'et', x2)
```

Types de données

- Type **numérique**

Entier (int), flottant (float) et complexe (complex)

- Type **chaine de caractères** (str)

Délimitée par des guillemets simples (') ou doubles (")

- Type **booléen** (bool)

Deux valeurs possibles True et False

```
1 i = 12
2 f = 25.99
3 c = 3 - 2j
4 s = "Hello World!"
5 b = True
```

Variable

- Une **variable** permet de stocker une valeur

Possède un nom et une valeur (d'un certain type)



- Deux opérations possibles
 - Initialisation
 - Affectation d'une valeur

```
1 var = 42
2 var = "¡ Holà amigos !"
```

Opérateur arithmétique

- Quatre **opérateurs arithmétiques** de base

Addition (+), soustraction (), multiplication () et division (/)*

- Opérateur d'**exponentiation** (**)

- Opération sur **nombres entiers**

Division entière (//) et reste de la division entière (%)

```
1 power = 2 ** 10          # 1024
2
3 # 17 = 5 * 3 + 2
4 quotient = 17 // 3       # 5
5 remainder = 17 % 3        # 2
```

Affectation composée

- **Combinaison** d'un opérateur arithmétique et d'affectation

Sucre syntaxique pour simplifier l'écriture du code

- `var = var ◊ value` \equiv `var ◊= value`

```
1 a += 1                      # a = a + 1
2
3 a **= 2                      # a = a ** 2
```

Priorité et associativité

- Ordre de **priorité** entre les opérateurs

Permet de définir l'ordre d'évaluation

- **Associativité** des opérateurs

Définit l'ordre d'évaluation en cas de même priorité

- Expression **complètement parenthésée**

Rend l'ordre d'évaluation explicite

```
1 a = 17 + 5 ** 2 * 2          # équivalent à (17 + ((5 ** 2) * 2))
2
3 b = 1 - 2 - 3                # équivalent à ((1 - 2) - 3)
```

Chaine de caractères

- Séquence d'échappement pour caractère spécial

Backslash (\\"), retour à la ligne (\n), tabulation (\t)...

- Opérateur de concaténation (+)
- Conversion en une chaîne de caractères avec la fonction str

```
1 name = "Cédric Marchand"
2 year = 1977
3
4 print(name + " n'est pas né en " + str(year) + ".")
```

Cédric Marchand n'est pas né en 1977.

Fonction print

- Fonction print pour **afficher du texte**

Affichage vers la sortie standard

- Deux paramètres nommés

- end indique le caractère à affiche en **fin de ligne** (\n)
- sep indique le caractère **séparateur** ()

```
1 day = 4
2 month = 8
3 year = 1961
4
5 print('Né le :', end=' ')
6 print(day, month, year, sep='/')
```

Né le : 4/8/1961

Fonction input

- Fonction input pour lire du texte

Lecture depuis l'entrée standard

- Conversion nécessaire pour lire d'autres types de données

```
1 serial = input('Bonjour, quel est ton matricule ? ')
2
3 print(serial, ", Président de l'ECAM !", sep='')
```

```
Bonjour, quel est ton matricule ? MLT
MLT, Président de l'ECAM !
```

Module et importation

- Importation d'un **module** avec `import`

Accès aux fonctions du module via son nom

- Importation des **fonctions d'un module** avec `from/import`

Accès direct aux fonctions du module

```
1 import cmath
2 from math import sin, radians
3
4 print("Racine carrée de -2 : ", cmath.sqrt(-2))
5 print("Sinus de 90 degrés : ", sin(radians(90)))
```

```
Racine carrée de -2 : 1.4142135623730951j
Sinus de 90 degrés : 1.0
```

Opérateur de comparaison

- Comparaison de deux valeurs
 - Égalité (==) et différence (!=)
 - Strictement plus grand/petit (>, <)
 - Plus grand/petit ou égal (>=, <=)
- Comparaison limitée aux types compatibles

Sans quoi il faut procéder à des conversions

```
1 a = 12 == 3 * 4      # a vaut True
2 b = "Eat" > "Drink"  # b vaut True
3 c = a != b           # c vaut False
```

Opérateur logique

- Combinaison d'expressions booléennes
 - NON logique (not) inverse une valeur
 - ET logique (and) impose les deux expressions à True
 - OU logique (or) nécessite une seule expression à True

a	b	not a	a and b	a or b
False	False	True	False	False
False	True		False	True
True	False	True	False	True
True	True		True	True

```
1 a = 8 > 2 and 12 <= 4           # a vaut False
2 b = 5 != 5 or 'PY' == 'P' + 'Y'    # b vaut True
```

Instruction if

- Exécution d'un bloc de code si une **condition est vérifiée**

Dans tous les cas, le programme continue après l'instruction if

```
1 x = -5
2 if x <= 0:
3     print("x est négatif !")
4     print("sa valeur absolue vaut ", -x)
5
6 print("Fin du programme")
```

```
x est négatif !
sa valeur absolue vaut 5
Fin du programme
```

Instruction if-else

- Exécution alternative si la condition n'est pas vérifiée

Dans tous les cas, le programme continue après le if-else

```
1 grade = 9.5
2 if grade >= 10:
3     print("vous avez réussi")
4 else:
5     print("vous avez raté")
6
7 print("Fin du programme")
```

```
vous avez raté
Fin du programme
```

Instruction if-elif-else

- Définition de plusieurs alternatives disjointes

Dans tous les cas, le programme continue après le if-elif-else

```
1 temp = 126
2 if temp < 100:
3     print("tout va bien")
4 elif 100 <= temp <= 130:
5     print("attention")
6 else:
7     print("danger")
8
9 print("Fin du programme")
```

```
attention
Fin du programme
```

Instruction while

- Répète un bloc de code tant qu'une condition est vérifiée

Dans tous les cas, le programme continue après while

```
1 n = 1
2 while n <= 5:
3     print(n)
4     n = n + 1
```

```
1
2
3
4
5
```

Définition de fonction

- Bloc de code **réutilisable et configurable**

Une fois définie, une fonction peut être appelée plusieurs fois

- Permet de structurer un programme et le rendre lisible

Moins de bug, travail collaboratif

```
1 # Définition de la fonction
2 def table7():
3     n = 1
4     while n <= 10:
5         print(n, "x 7 =", n * 7)
6         n += 1
7
8 # Appel de la fonction
9 table7()
```

Paramètre

- Fournir une valeur à la fonction grâce à un **paramètre**

Une paramètre est une variable définie dans la fonction

- On spécifie une **valeur au paramètre** au moment de l'appel

La valeur est affectée au paramètre dans la fonction

```
1 # Définition de la fonction
2 def table(base):
3     n = 1
4     while n <= 10:
5         print(n, "x", base, "=", n * base)
6         n += 1
7
8 # Appels de la fonction
9 table(7)
10 table(8)
11 table(42)
```

Valeur par défaut

- Valeur par défaut définie avec la fonction

Valeur du paramètre si elle n'est pas spécifiée lors de l'appel

- Utilisation des paramètres nommés lors de l'appel

```
1 # Définition de la fonction
2 def table(base, start=1, length=10):
3     n = start
4     while n < start + length:
5         print(n, "x", base, "=", n * base)
6         n += 1
7
8 # Appels de la fonction
9 table(8, 5, 2)
10 table(8)
11 table(8, 5)
12 table(8, length=2)
```

Valeur de retour

- Valeur de retour d'une fonction

La fonction génère une valeur lors de son appel

- Instruction return pour définir la valeur renvoyée

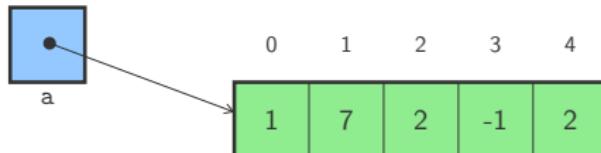
Définit la valeur de l'appel de fonction comme expression

```
1 # Définition de la fonction
2 def multiply(a, b):
3     return a * b
4
5 # Appels de la fonction
6 res = multiply(7, 9)
7 print(res)                  # 63
```

Liste

- Une liste stocke une séquence **ordonnée de valeurs**
 - Les éléments sont accessibles par leurs indices
 - La taille est accessible avec la fonction `len`

```
1 a = [1, 7, 2, -1, 2]
2
3 print(len(a))           # 5
```



Opérations sur les listes

■ Plusieurs opérations sur les listes

Accès ([]), suppression (del), concaténation (+), répétition (*)
Appartenance (in), comparaison (==, !=, <, <=, >, >=)

```
1 a = [1, 7, 2, -1, 2]
2
3 print(a[0])          # 1
4 print(a[-1])         # 2
5 print(7 in a)        # True
6 print(-1 not in a)   # False
7
8 b = [-9, 0]
9 print(b * 2)          # [-9, 0, -9, 0]
10 print(a + b)         # [1, 7, 2, -1, 2, -9, 0]
11 print(b == [-9, 0])   # True
```

Parcours d'une liste

- Parcours avec une boucle while et un compteur

Faire varier une variable de 0 à la taille de la liste moins un

- Parcours avec une **boucle for** et l'opérateur in

```
1 # Avec une boucle while
2 i = 0
3 while i < len(numbers):
4     print(numbers[i])
5     i += 1
6
7 # Avec une boucle for
8 for n in numbers:
9     print(n)
```

Modification d'une liste

- **Modification d'un élément** avec l'opérateur d'affectation

Les crochets permettent de sélectionner l'élément à modifier

- La fonction `del` permet de **supprimer un élément** d'une liste

Les autres éléments de la liste sont décalés

```
1 a = [1, 7, 2, -1, 2]
2
3 a[0] = 42                      # [42, 7, 2, -1, 2]
4 del(a[2])                      # [42, 7, -1, 2]
```

Slicing

- Extraire une sous-liste à partir d'une liste

On spécifie l'indice de départ (inclus) et d'arrivée (non inclus)

- Le slicing fait une **copie des éléments** de la sous-liste extraite

Mais il permet de modifier une liste lorsqu'utilisé à gauche de =

```
1 a = [1, 2, 6]
2
3 a[:0] = [0]                      # a = [0, 1, 2, 6]
4 a[4:] = [7, 8]                    # a = [0, 1, 2, 6, 7, 8]
5 a[3:3] = [3, 4, 5]                # a = [0, 1, 2, 3, 4, 5, 6, 7, 8]
6 a[0:4] = []                      # a = [4, 5, 6, 7, 8]
```

Chaine de caractères et intervalles

- Il existe des séquences **non modifiables**

Chaine de caractères et intervalles, par exemple

- **Intervalle d'entiers** délimités par deux bornes

Création avec la fonction range

```
1 s = "Hello"  
2 print(s[1:4])          # ell  
3  
4 i = range(5, 10)       # de 5 à 10 (non inclus)  
5 print(len(i))          # 5  
6 print(i[2])            # 7
```

Définition par compréhension

- La boucle `for` permet d'**itérer sur les éléments** d'une séquence

Définition par compréhension de séquences

```
1 # Boucle while
2 squares = []
3 i = 0
4 while i <= 100:
5     squares.append(i ** 2)
6     i += 1
7
8 # Boucle for
9 squares = []
10 for i in range(101):
11     squares.append(i ** 2)
12
13 # Définition par compréhension
14 squares = [i ** 2 for i in range(101)]
```

Ensemble

Giddy Up
COFFEE

GRILL MENU

BREAKFAST EDITION

BaGs

E G G
£ 2.50

MUSHROOM
+ - .50

E + B
£ 3.50

E + S
£ 3.50

BACON
£ 3

E
B
S
£ 4
B + S
3.50

SAUSAGE
£ 3

LUNCH

HOMEMADE BURGERS

Ensemble

- Collection **non ordonnée** d'éléments **distincts**

Pas de doublons et pas d'ordre entre les éléments

- Définition d'un ensemble avec {}

Opérations des séquences (sauf modification) applicables

```
1 numbers = {42, -2, 0, 7, 11}
2
3 print(numbers)           # {0, 42, 11, -2, 7}
4 print(len(numbers))      # 5
5 print(60 in numbers)    # False
6 print(type(numbers))     # <class 'set'>
7
8 for element in numbers:
9     print(element)
```

Définition

- Par **compréhension** ou à partir d'une **séquence**

Élimination automatique des doublons

- Ensemble vide avec `set()`

```
1 # {0, 42, 84, 21, 63}
2 S = {n for n in range(100) if n % 3 == 0 and n % 7 == 0}
3
4 # {0, 42, 12, -1}
5 A = set([12, 42, 0, 12, 0, -1, 0])
6
7 # {'!', 'C', 'i', 'c', 'o', 'r'}
8 B = set('Cocorico!')
```

$$S = \left\{ n \in \mathbb{N} \text{ avec } 0 \leq n < 100 \mid n \text{ est divisible par 3 et 7} \right\}$$

Modification d'un ensemble

- **Modification** d'un ensemble par l'ajout/suppression d'éléments

Utilisation des fonctions add et remove

- **Application de la fonction** sur la variable contenant la liste

*Nom de la variable, suivi d'un point (.) puis
du nom de la fonction à appeler, avec ses éventuels paramètres*

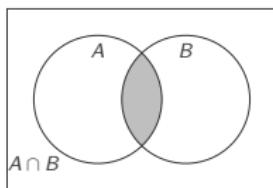
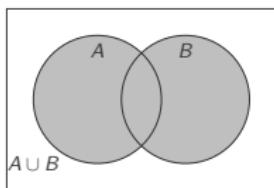
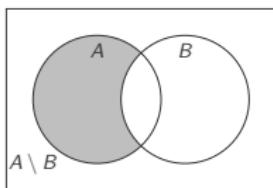
```
1 S = {1, 2, 3, 4}
2
3 S.remove(1)           # {2, 3, 4}
4 S.remove(2)           # {3, 4}
5 S.add(5)              # {3, 4, 5}
```

Opération ensembliste

- Trois **opérations ensemblistes** de base

Différence, union et intersection d'ensembles

- Ces fonctions créent toutes des **nouveaux ensembles**



```
1 A = {1, 2, 3, 4}
2 B = {3, 4, 5}
3
4 print(A - B)                  # {1, 2}
5 print(A & B)                  # {3, 4}
6 print(A | B)                  # {1, 2, 3, 4, 5}
```

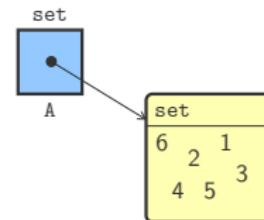
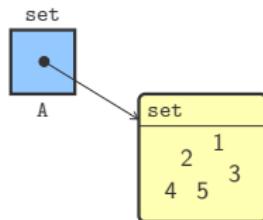
Comparaison add et | (1)

- La fonction add et l'opérateur | sont **similaires**

Tous les deux permettent d'ajouter un élément à un ensemble

- La fonction add ajoute un élément à l'ensemble

```
1 A = {1, 2, 3, 4, 5}           # {1, 2, 3, 4, 5}
2 A.add(6)                      # {1, 2, 3, 4, 5, 6}
```



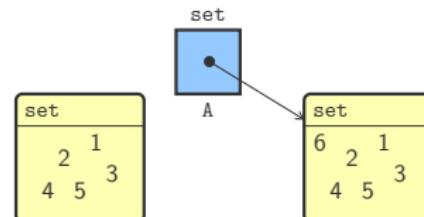
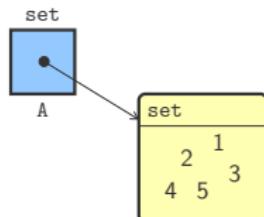
Comparaison add et | (2)

- La fonction add et l'opérateur | sont **similaires**

Tous les deux permettent d'ajouter un élément à un ensemble

- La fonction | crée un nouvel ensemble avec un élément ajouté

```
1 A = {1, 2, 3, 4, 5}
2 A |= {6}                      # {1, 2, 3, 4, 5, 6}
```



Éléments d'un ensemble

- Les éléments d'un ensemble doivent être **uniques**

*Par conséquent, ils doivent être **non modifiable***

```
1 # L'ensemble des sous-ensembles de {1, 2, 3}
2 A = [{}, {1}, {2}, {3}, {1, 2}, {1, 3}, {2, 3}, {1, 2, 3}]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: unhashable type: 'set'
```

Le type frozenset

- Le type frozenset représente un ensemble non modifiable

Les opérations des ensembles, sauf modification, sont applicables

```
1 # L'ensemble des sous-ensembles de {1, 2, 3}
2 A = {
3     frozenset(),
4     frozenset({1}), frozenset({2}), frozenset({3}),
5     frozenset({1, 2}), frozenset({1, 3}), frozenset({2, 3}),
6     frozenset({1, 2, 3})
7 }
```

Somme des éléments d'un ensemble

- On déclare une variable `sum` pour la somme
- On parcourt les éléments de l'ensemble en mettant `sum` à jour
- On renvoie la somme

```
1 def sumElements(S):
2     sum = 0
3     for elem in S:
4         sum += elem
5     return sum
6
7 S = {1, 2, 3}
8 print(sumElements(S)) # 6
```

Plus grande valeur d'un ensemble

- On déclare une variable `max` pour la plus grande valeur trouvée
- On parcourt les éléments de l'ensemble en mettant `max` à jour
- On renvoie la plus grande valeur trouvée

```
1 def maxElement(S):
2     max = float("-inf")
3     for elem in S:
4         if elem > max:
5             max = elem
6     return max
7
8 S = {1, 24, -2, 99, 16}
9 print(maxElement(S)) # 99
```

Dictionnaire



Dictionnaire

- Ensemble de paires clé-valeur

Les clés sont uniques et non modifiables

- Définition d'un dictionnaire avec {}

Opérations des séquences (sauf indices) applicables

```
1 phone = {'Quentin': 8723, 'Cédric': 2837, 'Nathalie': 4872}
2
3 print(phone)           # {'Quentin': 8723, 'Cédric': 2837,
4                           #   'Nathalie': 4872}
5 print(len(phone))      # 3
6 print('Cédric' in phone) # True
7 print(type(phone))     # <class 'dict'>
```

Définition

- Par compréhension ou à partir d'une liste de paires clé-valeur

Élimination automatique des doublons

- Dictionnaire vide avec {}

```
1 # {1: 1, 3: 9, 9: 81, 5: 25, 7: 49}
2 square = {n : n ** 2 for n in range(1,10) if n % 2 != 0}
3
4 # {'A': 65, 'C': 67, 'B': 66, 'E': 69, 'D': 68, 'F': 70}
5 mapping = {chr(i): i for i in range(65, 71)}
```

Accès et modification

- Accès à une valeur à l'aide de la clé entre crochets

Permet également la modification d'une valeur

- Suppression avec la fonction del

- Deux situations si la clé n'existe pas

- À droite de = provoque une erreur
- À gauche de = ajoute une paire au dictionnaire

```
1 price = {"lemon": 0.85, "pear": 1}
2
3 price['lemon'] = 0.90      # {"lemon": 0.90, "pear": 1}
4 price['apple'] = 1.00      # {"lemon": 0.90, "pear": 1, "apple": 1}
5 del(price['pear'])        # {"lemon": 0.90, "apple": 1}
```

Parcours d'un dictionnaire

- Accès aux clés avec la fonction `keys` et aux paires avec `items`

Renvoient des séquences qu'on peut convertir en liste

```
1 # ['lemon', 'pear', 'apple']
2 print(list(price.keys()))
3
4 # [('lemon', 0.9), ('pear', 1.0), ('apple', 1.0)]
5 print(list(price.items()))
6
7 # Parcours avec les clés
8 for fruit in price.keys():
9     print(fruit, price[fruit], sep=' : ')
10
11 # Parcours direct
12 for key, value in price.items():
13     print(key, value, sep=' : ')
```

Somme des durées

- Liste de musiques dont il faut calculer la durée
- Chaque musique est stockée avec un dictionnaire

```
1 def totalDuration(playlist):
2     duration = 0
3     for music in playlist:
4         duration += music['duration']
5     return duration
6
7 myPlaylist = [
8 {'artist': 'Axwell /\ Ingrosso', 'title': 'Sun Is Shining', 'duration': 250},
9 {'artist': 'Black M', 'title': 'Sur ma route', 'duration': 251},
10 {'artist': 'AronChupa', 'title': "I'm an Albatraoz", 'duration': 167}
11 ]
12 print(totalDuration(myPlaylist)) # 668
```

Liste des titres interprétés par un artiste

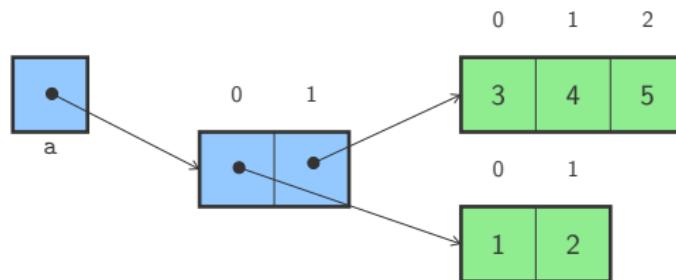
- Parcours du dictionnaire, à la recherche de l'artiste
- Construction d'une liste des titres

```
1 def findTitles(playlist, artist):
2     titles = []
3     for music in playlist:
4         if music['artist'] == artist:
5             titles.append(music['title'])
6     return titles
7
8 myPlaylist = [
9 {'artist': 'Axwell /\ Ingrosso', 'title': 'Sun Is Shining', 'duration': 250},
10 {'artist': 'Black M', 'title': 'Sur ma route', 'duration': 251},
11 {'artist': 'AronChupa', 'title': "I'm an Albatraoz", 'duration': 167}
12 ]
13 print(findTitles(myPlaylist, "AronChupa")) # ["I'm an Albatraoz"]
```

Imbrication de listes

- On peut construire une liste dont les éléments sont des listes
- Accès multiples à faire pour accéder aux éléments imbriqués

```
1 A = [1, 2]
2 B = [3, 4, 5]
3 L = [A, B]                      # Équivalent à L = [[1, 2], [3, 4, 5]]
4
5 print(L[0][1])                  # 2
6 print(L[1][2])                  # 5
```



Imbrication de structures de données

- On peut imbriquer des séquences, ensembles et dictionnaires

```
1 address = {'street': "Promenade de l'Alma", 'number': 50, 'zip':  
2 1200, 'city': "Woluwé-Saint-Lambert"}  
3 marchand = {'firstname': "Cédric", 'lastname': "Marchand", 'address':  
4 address}  
5  
# Équivalent à  
6 # marchand = {'firstname': "Cédric", 'lastname': "Marchand", '  
7 address': {'street': "Promenade de l'Alma", 'number': 50, 'zip':  
8 1200, 'city': "Woluwé-Saint-Lambert"}}  
9  
print(marchand['firstname'])           # Cédric  
print(marchand['address']['city'])     # Woluwé-Saint-Lambert
```

Copie

- Affecter une même liste à deux variables crée un **alias**
Même chose pour les séquences, les ensembles et les dictionnaires
- On crée une **véritable copie** de liste avec la fonction `list`
Ou avec les fonctions `set`, `dict`...

```
1 L = [1, 2, 3]                                     # A est un alias de L
2 A = L
3 A[0] = 42
4 print(L)                                         # [42, 2, 3]
5
6 L = [1, 2, 3]                                     # B est une copie de L
7 B = list(L)
8 B[0] = 42
9 print(L)                                         # [1, 2, 3]
```

Copie de structures imbriquées

- Pas de soucis de copies pour les collections non modifiables
- La copie ne se fait pas en profondeur

Seuls les éléments de « premier niveau » sont copiés

```
1 L = [[1], [2, 3], [4, 5, 6]]                      # A est une copie de L
2 A = list(L)
3
4 A[2] = [42]
5 print(L)                                         # [[1], [2, 3], [4, 5, 6]]
6
7 A[1][0] = 42
8 print(L)                                         # [[1], [42, 3], [4, 5, 6]]
9
10 A[2][0] = 999
11 print(L)                                         # [[1], [42, 3], [4, 5, 6]]
12                                              # Pourquoi pas
13                                              # [[1], [42, 3], [999, 5, 6]] ?
```

Module copy

- Deux fonctions proposées par le **module copy**
 - `copy` pour une copie « *shallow* »
 - `deepcopy` pour une copie « *deep* »
- Une copie en profondeur peut prendre du temps

Et aussi consommer beaucoup d'espace mémoire

```
1 import copy
2
3 L = [[1], [2, 3], [4, 5, 6]]
4 A = copy.copy(L)                      # A est une copie shallow de L
5 B = copy.deepcopy(                     # A est une copie deep de L
```



Base de données

Base de données

- Collection de données organisées selon un format choisi

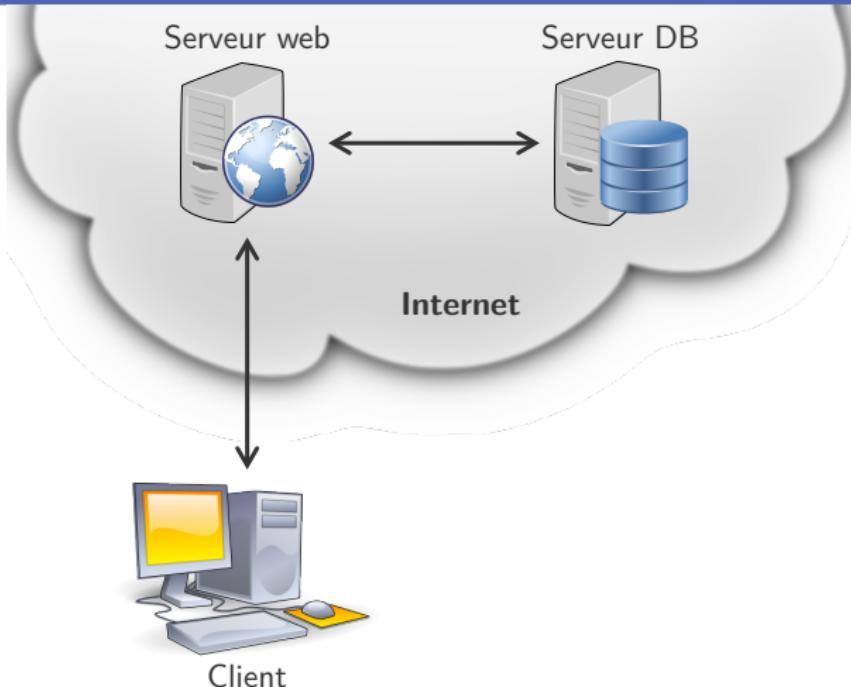
Similitudes avec les dictionnaires Python

- Système de gestion de bases de données (SGBD)

Ensemble de logiciels qui gère la base de données

- Interrogation de la base de données
- Gestion du stockage des données
- ...

Accès à un site web



JavaScript Object Notation (JSON) (1)

- Permet de **représenter des objets**

Notation issue de la notation des objets Javascript

- Ensemble de **paires** (étiquette, valeur)

Étiquette entre guillemets, deux-points et valeur

- **Valeurs** sont soit une simple valeur, soit une liste de valeurs

Liste de valeurs délimitée par des crochets []

JavaScript Object Notation (JSON) (2)

```
1  {
2      "name": "Carnet d'adresses de Sébastien Combéfis",
3      "contacts": [
4          {
5              "firstname": "Cédric",
6              "lastname": "Marchand",
7              "phone": 2837
8          },
9          {
10             "firstname": "Nathalie",
11             "lastname": "Siebert",
12             "phone": 4872
13         },
14         {
15             "firstname": "Quentin",
16             "lastname": "Lurkin",
17             "phone": 8723
18         }
19     ]
20 }
```

Base de données orientée document

- Stocke des **documents JSON**

Souplesse par l'absence de schéma prédéfini

- Données **répartissables** sur plusieurs machines
- **Sérialisation et désérialisation** des documents JSON

Vers des objets du langage cible, pour être utilisés



Sérialisation

- Sérialisation d'un dictionnaire en document JSON
- Fonction `dumps` du module json

Renvoie une chaîne de caractères

```
1 import json
2
3 bb = {'seasons': 5, 'genre': ['crime drama', 'thriller']}
4 skins = {'seasons': 7, 'genre': ['teen drama', 'comedy drama']}
5 tvshows = {'Breaking Bad': bb, 'Skins': skins}
6
7 document = json.dumps(tvshows, indent=4)
```

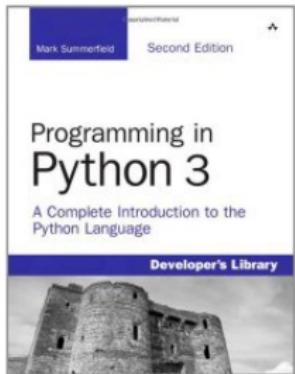
Désérialisation

- Désérialisation d'un document JSON en dictionnaire
- Fonction `loads` du module `json`

Renvoie un dictionnaire

```
1 import json
2
3 document = '{"Belgium":{"capital":"Brussels","languages":["french",
4   "dutch","german"]},"China":{"capital":"Beijing","languages":[]>
5   ["mandarin","chinese"]}}'
countries = json.loads(document)
```

Livres de référence



ISBN

978-0-321-68056-3



ISBN

978-2-212-13434-6

Crédits

- Photos des livres depuis Amazon
- <https://www.flickr.com/photos/sharynmorrow/14549114>
- <https://www.flickr.com/photos/doctorow/11082104723>
- <https://www.flickr.com/photos/greeble/3338710223>
- <https://www.flickr.com/photos/shindotv/3835365695>
- <https://openclipart.org/detail/25319/cartoon-cloud>
- <https://openclipart.org/detail/17924/computer>
- <https://openclipart.org/detail/163741/web-server>
- <https://openclipart.org/detail/163711/database-server>
- https://commons.wikimedia.org/wiki/File:MongoDB_Logo.png
- <https://svn.apache.org/repos/asf/couchdb/supplement/logo/couchdb-logo.png>
- <https://commons.wikimedia.org/wiki/File:Riaklogo.png>
- https://en.wikipedia.org/wiki/File:Redis_Logo.svg