

LINGE1225 : Programmation en économie et gestion

Point Théorique 6

Dictionnaires, lecture et écriture de fichier, et exceptions

François Fouss & Marco Saelens

Année académique 2020-2021

1

Livre de référence

- Chapitre 9 : Manipuler des fichiers
- Chapitre 10 : Approfondir les structures de données



2

Plan

- Dictionnaires
- Application dictionnaires
- Lecture/écriture d'un fichier « .csv »*
- Exceptions
- Application fichier et exceptions

* *Comma-separated values*, connu sous le sigle CSV, est un format texte ouvert représentant des données tabulaires sous forme de valeurs séparées par des virgules.
https://fr.wikipedia.org/wiki/Comma-separated_values

3

Dictionnaires

4

Dictionnaire

- Les dictionnaires sont modifiables comme les listes mais ne sont pas des séquences.
- Les éléments enregistrés ne sont pas disposés dans un ordre immuable.
- Il est possible d'accéder à n'importe quel élément à l'aide d'un index spécifique appelé **clé**. Pas comme avec des indices ds les listes (≠ séquencé)
- Les éléments mémorisés dans un dictionnaire peuvent être de n'importe quel type : valeurs numériques, des chaînes, des listes, des dictionnaires et aussi des fonctions.

ex type dictionnaire : association d'infos à 1 pers

5

Dictionnaire Vs Liste

défini par des { }

Dictionnaire	Liste
<ul style="list-style-type: none"> • Syntaxe : <code><Variable> = {<Clé> : <Valeur>, ...}</code> • L'ordre des objets n'a pas d'importance • Utile pour trouver des informations associées à une clé • On trouve une valeur en connaissant sa clé • Les clés sont uniques pour chaque valeur • Mutable = modifiable <p>Pour 1 clé donnée = 1 seule valeur (semblable à 1 fct) MAIS on peut changer cette valeur (cad les valeurs sont modifiables/mutables)</p>	<ul style="list-style-type: none"> • Syntaxe : <code><Variable> = [<valeur>, <Valeur>, ...]</code> • L'ordre a de l'importance • Très utile lorsque la liste est triée • On trouve une valeur en connaissant son indice dans la liste • L'indice est unique pour chaque valeur • Mutable

6

Dictionnaire : exemple

- Voici un exemple de dictionnaire
- Ce dictionnaire comprend la traduction de mots de l'anglais au français,

#Dictionnaire

Clé	Valeur
'computer'	'ordinateur'
'keyboard'	'clavier'
'mouse'	'souris'

```
dico = {'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

Les clés sont uniques, pas les valeurs !

- Une clé est associée une valeur MAIS une valeur peut avoir plusieurs clés associées.

ex: 2 mots en anglais peuvent avoir 1 même trad

- 1) to walk = marcher & the market = le marché → 2 clés ≠ mais 1 seule valeur pour les 2 marché
- 2) nice = gentil & friendly = gentil → 2 clés ≠ mais 1 seule et même valeur pour les 2

7

Création Dictionnaires

- Exemple : dictionnaire de langue Français - Anglais

Nom
du
dictionnaire

création d'un dictionnaire vide grâce à une
paire d'accolades

Même chose que ci-dessus
avec une syntaxe ≠

```
#Dictionnaire
dico = {}
dico['computer'] = 'ordinateur'
dico['mouse'] = 'souris'
dico['keyboard'] = 'clavier'
print(dico)

{'computer': 'ordinateur', 'keyboard': 'clavier', 'mouse': 'souris'}
```

création des éléments
du dictionnaire

clé

Dictionnaire : traduction
français/anglais

8

Création Dictionnaires : clés

- Dans les dictionnaires, les **index s'appellent donc des clés**, et les éléments peuvent donc s'appeler des **paires clé-valeur**. Dans le dictionnaire anglais français, les clés et valeurs sont des chaînes de caractères.

Important : l'ordre dans lequel on fournit les informations au dictionnaire est différent de l'ordre dans lequel les éléments apparaissent dans le dictionnaire. C'est pourquoi on appelle les valeurs d'un dictionnaire grâce à sa clé tel que :

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

Supprimer un élément d'un dictionnaire :

```
del dico['mouse']
```

Connaître la longueur du dictionnaire :

```
print(len(dico))
```

9

Test d'appartenance : *in*

Pour vérifier si 1 valeur se trouve dans le dico

```
print(Dico)

{'computer': 'ordinateur', 'keyboard': 'clavier'}

if 'mouse' in Dico:
    print('OK')
else:
    print('KO')

KO
```

Nous venons de supprimer 'mouse' du dictionnaire, il ne s'y trouve donc plus.

« in » permet de renvoyer vrai si la clé se trouve ds le dico et faux si la clé ne se trouve pas ds le dico

10

Parcours d'un dictionnaire

#affichage des clés (similaire aux listes)

```
for clef in dico:
    print(clef)
```

```
keyboard
computer
```

Pour effectuer une boucle sur tt les clés dispo

clef va énumérer tt les
associations clé-valeur →

#affichage des clés et valeurs

```
for clef,valeurs in dico.items:
    print(clef,valeurs)
```

```
keyboard, clavier
computer, ordinateur
```

Pour énumérer tt les associations clé-valeur

11

Dictionnaires : utilité?

- Du fait qu'ils ne sont pas des séquences, les dictionnaires sont utiles pour gérer des ensembles de données où l'on est amené à effectuer fréquemment des ajouts ou suppressions, dans n'importe quel ordre.
- Ils remplacent les listes lorsqu'il s'agit de traiter des ensembles de données numérotées dont les numéros d'indice ne se suivent pas.

=> Les clés peuvent être des strings, des entiers, des nombres décimaux

#dictionnaire : utilité

```
client = {}
client[4337] = 'Dupond'
client[256] = 'Durant'
client[743] = 'Schmidt'
```

Ici on associe le nom du client
à chaque numéro de client

12

Application dictionnaires

13

Exercice

Ecrivez une fonction qui échange les clés et les valeurs d'un dictionnaire (ce qui permettra par exemple de transformer un dictionnaire anglais/français en un dictionnaire français/anglais). On suppose que le dictionnaire ne contient pas plusieurs valeurs identiques.

➤ Solution

```
#Echange des clés et des valeurs dans un dictionnaire

def inverse(dico):
    "Construction d'un nouveau dico, pas à pas"
    dico_inv={} → initialise à un dico vide
    for cle in dico:
        item=dico[cle]
        dico_inv[item] = cle item devient la clé de dico_inv & cle devient la valeur de dico_inv

    return dico_inv
```

➤ Programme test

```
#Echange des clés et des valeurs dans un dictionnaire

dico = {'computer' : 'ordinateur',
        'mouse' : 'souris',
        'keyboard' : 'clavier',
        'Hard disk' : 'disque dur',
        'Screen' : 'écran'}

print(dico)
print(inverse(dico))
```

14

Il faut que la relation soit univoque cad par cle 1 seule valeur (car sinon on aura plusieurs clés les mêmes)

Lecture/écriture de fichiers « .csv »

15

Qu'est-ce qu'un fichier .csv ?

- Un **fichier .csv** (comma-separated values) est un fichier qui représente des données sous forme d'un tableau où les valeurs sont séparées par une virgule (représente les colonnes) et un retour à la ligne pour représenter une ligne :

Test.csv

```
1 0,5,9
2 7,6,8
3 3,4,5
4 6,1,7
5
```

Équivalent en tableau

0	5	9
7	6	8
3	4	5
6	1	7

Excel peut être utilisé si vous n'avez pas d'éditeur de texte (vous devez juste nommer votre fichier « .csv » à la fin et non « .xlsx »)

	A
1	0,5,9
2	7,6,8
3	3,4,5
4	6,1,7
5	

On souhaite sauvegarder les infos

!!! Word n'est pas un éditeur de texte pur

16

Ouvrir un fichier

- Sous Python, l'accès aux fichiers est assuré par l'intermédiaire d'un objet-interface particulier, que l'on appelle **objet-fichier**.
- La création de cet objet se fait à l'aide de la fonction intégrée **open()**.
- Cette fonction renvoie un objet ayant des méthodes (fonctionnalités) spécifiques, qui vous permettront de lire et/ou écrire dans le fichier.

17

Ecrire un fichier

objet-fichier fichier véritable (sur disque ou disquette)

```
#Ecriture séquentielle
obFichier = open('Monfichier.txt', 'a')
obFichier.write('Bonjour, fichier !\n')
obFichier.write("Quel beau temps, aujourd'hui!\n")
obFichier.close()
```

À la ligne

On renferme le fichier, permet d'arrêter le flux

- Création de l'objet-fichier obFichier, qui fait référence à un fichier véritable dont le nom sera Monfichier.
- La fonction open() attend **deux arguments** de type chaînes de caractères. Le premier est le nom du fichier à ouvrir et le deuxième la méthode d'ouverture.
- Ici le mode d'ouverture "a" = il faut ouvrir ce fichier en mode "ajout". C'est comme si on faisait « append » : on rajoute à la fin du fichier
- Il existe un autre mode "w", qui crée toujours un nouveau fichier lorsqu'on ajoute des données (il « overwrite ») contrairement à la méthode "a" qui ajoute les données à la fin du fichier. On supprime l'ancien fichier et on remplace par les nouvelles données

18

C'est un flux qui va de l'exécution du programme vers un emplacement du disque dur

Lire un fichier

- Une fois ouvert en mode « lecture », on peut lire le contenu ligne par ligne.
- Ensuite, parcourir le contenu ligne par ligne à l'aide d'une boucle *for*.

#Lecture séquentielle

```

ofi = open('Monfichier.txt', 'r')
t = ofi.readlines()
for line in t :
    print(line)
ofi.close()

```

Nom du fichier à ouvrir

Ouverture du fichier en mode « lecture »

Affiche ligne par ligne le contenu du fichier « fichier.txt »

referme le fichier après usage

Bonjour, fichier !
Quel beau temps, aujourd'hui !

- Il existe aussi la méthode `read()` qui lit les données présentes dans le fichier et les transfère dans une variable de type chaîne de caractères (string).

19

Lire un fichier .csv

- Python propose une bibliothèque pour la manipulation de fichiers .csv.
- Comme vu précédemment, pour utiliser des « modules » en python, il faut mettre en tête de page : `import <module>`.
- Le module pour les fichiers .csv est : `csv`
- Pour lire un fichier, on utilise la méthode « `reader(<fichier>)` ».

Exemple :

Contenu du fichier .csv :

```

Test.csv
1 0,5,9
2 7,6,8
3 3,4,5
4 6,1,7

```

Code :

```

import csv
f = open("Test.csv", "r")
fichier = csv.reader(f)
for line in fichier:
    print(line)
f.close()

```

Résultat :

```

['0', '5', '9']
['7', '6', '8']
['3', '4', '5']
['6', '1', '7']

```

Chaque ligne du fichier est transformée en une liste de string

20

Ecrire un fichier .csv

- Maintenant que nous avons vu comment lire un fichier .csv, en écrire un n'est pas plus compliqué !
- Il suffit de créer une liste à deux dimensions et de l'écrire dans le fichier comme on le souhaite et avec le symbole de séparation que l'on veut (par défaut une virgule).
- Pour écrire, on utilise l'instruction `<fichier>.writerows(<liste>)`.

Exemple :

```
import csv
f = open("Test.csv", "w", newline = " ")
lst = [["1", "2", "3"], ["8", "9", "7"]]
fichier = csv.writer(f)
fichier.writerows(lst)
f.close()
```

Sans cette instruction, il y aura un saut de 2 lignes entre chaque ligne que vous écrivez ...

Résultat dans le fichier « Test.csv » lorsqu'on l'ouvre:

```
Test.csv
1 1,2,3
2 8,9,7
```

```
import csv
f = open("Test.csv", "w", newline = " ")
lst = [["1", "2", "3"], ["8", "9", "7"]]
fichier = csv.writer(f, delimiter = "A")
fichier.writerows(lst)
f.close()
```

Résultat dans le fichier « Test.csv » lorsqu'on l'ouvre:

```
Test.csv
1 1A2A3
2 8A9A7
```

21

Exceptions

22

Exceptions

- Lorsqu'il y a une erreur dans un code, l'exécution est généralement interrompue et un message d'erreur plus ou moins explicite est affiché:

```
#Exception  
print(22/0)  
ZeroDivisionError: int division or modulo by zero
```

Type d'erreur

Information spécifique sur l'erreur

23

Exceptions

- Dans de nombreux cas, il est possible de prévoir à l'avance certaines des erreurs qui peuvent se produire.
- Il faut alors inclure dans le programme (à l'endroit où l'erreur pourrait se produire) des instructions particulières indiquant ce qu'il faut faire si l'erreur se produit.
- Ces instructions spécifiques ne seront jamais exécutées si l'erreur ne se produit pas.

24

Bloc try – except – else

- Python utilise les instructions *try - except - else* pour gérer les erreurs.
- Le bloc d'instructions qui suit directement une instruction **try** est exécuté *sous réserve*.
- Si une erreur survient pendant l'exécution de l'une de ces instructions, alors Python annule cette instruction et exécute à sa place le code inclus dans le bloc qui suit l'instruction **except**.
- Si aucune erreur traitée dans les instructions **except** ne s'est produite, alors c'est le bloc qui suit l'instruction **else** qui est exécuté.
- Dans tous les cas, l'exécution du programme peut se poursuivre ensuite avec les instructions ultérieures.

25

Exemple

```
#Bloc try - except - else
def existe(fname):
    try:
        f = open(fname, 'r')
        f.close()
        return True
    except:
        return False

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename):
    print("Ce fichier existe bel et bien.")
else:
    print("Le fichier", filename, "est introuvable.")
```

Si une erreur arrive, on va dans le « except ». Au sinon, le « except » n'est pas exécuté. Cela permet à ce que le code ne s'arrête pas en cas d'erreur

- Il est également possible de faire suivre l'instruction **try** de plusieurs blocs **except**, chacun d'entre eux traitant un type d'erreur spécifique (voir documentation Python).

26

Application fichiers et exceptions

27

Générer une matrice à partir d'un fichier .csv

Vous voulez créer une matrice à partir d'un fichier .csv. De plus, vous voulez des integers à la place de strings pour pouvoir faire des calculs par la suite. /!\ Vous devez gérer tous les cas d'erreur qui pourraient se produire.

Solution :

```
import csv

def get_a_list(fichier) :
    try :
        f = open(fichier, "r")
        fichier_ouvert = csv.reader(f)
        lst = []
        try :
            for line in fichier_ouvert:
                lst_bis = []
                for value in line :
                    value_bis = int(value)
                    lst_bis.append(value_bis)
                lst.append(lst_bis)
            f.close()
            return lst
        except :
            f.close()
            print ( "Valeur non « castable » en integer")
            return 0
    except :
        print ( "Votre fichier n'existe pas")
        return 0

ma_liste = get_a_list("Test.csv")
print (ma_liste)
```

28