

LINGE1225 : Programmation en économie et gestion

Cours 6

Dictionnaires, lecture et écriture de fichier et exceptions

François Fouss & Marco Saerens

Année académique 2020-2021

Livre de référence

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



Plan

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

Dictionnaire

4

Dictionnaire : exemple

- Voici un exemple de dictionnaire
- Ce dictionnaire comprend la traduction de mots de l'anglais vers le français

```
#Dictionnaire
```

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

5

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**.
- 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.

6

Dictionnaire Vs Liste

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 associé à une clé • On trouve une valeur en connaissant sa clé • Les clés sont uniques pour chaque valeur • Mutable 	<ul style="list-style-type: none"> • Syntaxe : <code><Variable> = [<valeur>, <Valeur>, ...]</code> • L'ordre a de l'importance • 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

7

Création Dictionnaire

➤ Exemple : dictionnaire de langue Français - Anglais

Nom du dictionnaire

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

On associe la clé « computer » à la valeur « ordinateur » dans le dictionnaire « dico »

création des éléments du dictionnaire

clé

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

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

Dictionnaire : traduction français/anglais

8

Création Dictionnaire

```
#Dictionnaire
print(dico)

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

Accolade

- Un dictionnaire dans la syntaxe Python est composé d'une série d'éléments séparés par des virgules, le tout étant enfermé entre deux accolades. Chacun de ces éléments est lui-même constitué d'une paire d'objets : un index et une valeur, séparés par un double point.

9

Création Dictionnaires : clés

- Dans les dictionnaires, les index s'appellent 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 ;

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

Nom du dictionnaire

10

Opérations sur les dictionnaires

➤ Exemple : dictionnaire d'un stock de fruit

#Dictionnaire

```
invent = {'pommes': 430, 'bananes' : 312, 'oranges' : 274,  
'poires' : 137}
```

```
print(invent)
```

```
{'oranges': 274, 'pommes': 430, 'bananes':312, 'poires':  
137}
```

```
print(invent['pommes'])
```

```
430
```

```
print(invent[430])
```

ERREUR : 430 n'est pas une clé ! On ne peut pas retrouver
une clé apd d'une valeur car chaque clé a une valeur unique
mais une valeur peut avoir plusieurs clés.
Exemple: invent = {'pomme' : 430, 'bananes' : 430}

11

Opérations sur les dictionnaires

➤ Comment enlever les pommes du dictionnaire ?

Utilisation de la méthode del

```
del invent['pommes']
```

```
print(invent)
```

```
{'orange': 274, 'bananes': 312, 'poires': 137}
```

12

Opérations sur les dictionnaires

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

```
#opération sur dictionnaire
```

```
print(len(invent))
```

```
3
```

13

Test d'appartenance

- L'instruction `in` est utilisable avec les dictionnaires. Elle permet de savoir si un dictionnaire comprend une clé bien déterminée :
- Reprenons l'exemple des fruits :

```
invent
{'oranges': 274, 'bananes': 312, 'poires': 137}

if 'pommes' in invent:
    print('Nous avons des pommes')
else:
    print('Pas de pommes. Sorry')

Pas de pommes. Sorry
```

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

14

Parcours d'un dictionnaire

➤ Comment parcourir un dictionnaire ?

```
invent = {'oranges':274, 'poires':137, 'bananes':312}  
for clef in invent:  
    print(clef)
```

poires
bananes
oranges

} Ordre différent que lors de la définition du dictionnaire

Remarques :

- 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* (puisque un dictionnaire n'est pas une séquence)

15

Parcours d'un dictionnaire

➤ Comment parcourir un dictionnaire ?

➤ Affichage des clés et valeurs

```
invent = {'oranges':274, 'poires':137, 'bananes':312}  
for clef,valeurs in invent.items():  
    print(clef,valeurs)
```

poires,137
bananes,312
oranges,274

utilisation de la
méthode `items` qui
retourne un tuple

16

Les clés

- Jusqu'à présent les clés des dictionnaires étaient toujours de type *string*
- En fait, il est possible d'utiliser en guise de clés n'importe quel type de données non modifiables : des entiers, des réels et des chaînes de caractères

17

Dictionnaires : séquences ?

- Les éléments d'un dictionnaire ne sont pas disposés dans un ordre particulier. Des opérations comme la concaténation et l'extraction ne peuvent pas s'appliquer dans ce cas ci.

Exemple

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

- Python vous affichera une erreur. Cette commande est impossible.

18

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 ne se suivent pas.

```
#dictionnaire : utilité
```

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

19

Application dictionnaire

20

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.

21

Exercice

➤ Solution

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

def inverse(dico):
    ↔ "Construction d'un nouveau dico, pas à pas"
    dic_inv={}
    for cle in dico:
        ↔ item=dico[cle]
        dic_inv[item] = cle

    ↔ return dico_inv
```

22

Exercice

➤ 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))
```

23

Exercice supplémentaire

24

Exercice

- Vous êtes banquier et imaginons que vous voulez créer un répertoire qui reprend le nom de vos clients, leurs prénoms ainsi que leur numéro de téléphone.
- Attention, pour cet exercice vous devrez créer une liste de dictionnaire reprenant ces informations pour chaque client.
- Ensuite, il vous est demandé de créer une fonction "trouve" qui affichera le numéro de téléphone et le prénom de la personne lorsque vous y rentrerez le nom de famille.

Signature : `def trouve (nom) :`

Les coordonnées de vos clients sont :

Dupont Michel 044433322
Dutunnel Julie 055544433
Dusousmarin Louise 088877766

25

Exercice : solution

```
lst1 = {'numero': '044433322', 'prenom': 'Michel', 'nom':  
        'Dupont'}  
lst2 = {'numero': '055544433', 'prenom': 'Julie', 'nom':  
        'Dutunnel'}  
lst3 = {'numero': '088877766', 'prenom': 'Louise', 'nom':  
        'Dusousmarin'}  
nv_lst = []  
nv_lst.append(lst1)  
nv_lst.append(lst2)  
nv_lst.append(lst3)  
  
print(nv_lst)
```

26

Exercice : solution

```
print(nv_lst[1]['prenom'])

def trouve(nom):
    ↔ for i in nv_lst:
    ↔     ↔ if nom in i['nom']:
    ↔     ↔     ↔ print(i['numero'], i['prenom'])

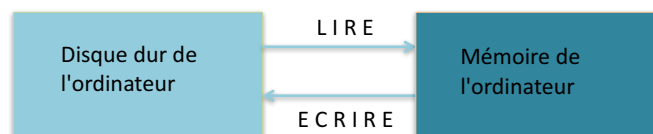
trouve('Dupont')
```

27

Lecture/écriture de fichiers

Utilité des fichiers

- Les fichiers sont utilisés lorsque nous souhaitons traiter une quantité d'informations plus importante
- Un fichier se compose de données enregistrées sur le disque dur, une clef USB ou un CD.
- L'accès est possible grâce à son nom



29

Utilité des fichiers – Disque dur vs RAM



- Utilisé pour écrire des fichiers
- Contient/conservé les informations si l'ordinateur est éteint
- Plus lent
- 128Go, 256Go, ...



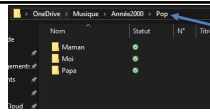
- Utilisé pour lire des fichiers
- Contient des informations que lorsque l'ordinateur est allumé
- Utilisé pour stocker tout les calculs/données qu'a besoin le processeur
- Plus rapide
- 8Go, 16Go, ...

Donc:

- un disque dur/ SSD est utile pour stocker des informations même lorsque l'ordinateur est éteint → on y écrit les fichiers
- La RAM est plus rapide et donc utile pour ouvrir rapidement des fichiers et stocker les informations temporairement → on l'utilise pour lire des fichiers.

30

Noms de fichiers



Nous sommes dans le répertoire courant « Pop » où se trouve 3 dossiers

- En général, les fichiers créés et/ou recherchés par Python se font dans le répertoire courant*.
- Il est possible de forcer Python à changer son répertoire courant
- Exemple : le répertoire visé est /home/jules/exercices.

```
#nom de fichier
from os import chdir
chdir("/home/jules/exercices")
```

Fonction pour changer de répertoire

module

- La première commande importe la fonction `chdir()` du module `os`
- Le module `os` contient toute une série de fonctions permettant de dialoguer avec le système d'exploitation.
- La seconde commande provoque le changement de répertoire

os = operating system
chdir = change directory

*Le répertoire courant = le dossier dans lequel vous vous trouvez avec python. Par exemple : quand vous êtes dans un dossier, vous ne voyez que les fichiers qu'il contient et non les fichiers précédents ou les fichiers des sous-dossiers. Donc Python ne fait que des recherches dans le « dossier » dans lequel il se trouve).

31

Ecriture séquentielle

- 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 de méthodes spécifiques, qui vous permettront de lire et écrire dans le fichier

32

Écriture séquentielle

objet-fichier

fichier véritable (sur disque ou disquette)

#Écriture séquentielle

```
obFichier = open('Monfichier', 'a')
obFichier.write('Bonjour, fichier !')
obFichier.write("Quel beau temps, aujourd'hui!")
obFichier.close()
```

- 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".
- 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.

33

Écriture séquentielle

objet-fichier

fichier véritable (sur disque ou disquette)

#Écriture séquentielle

```
obFichier = open('Monfichier', 'a')
obFichier.write('Bonjour, fichier !')
obFichier.write("Quel beau temps, aujourd'hui!")
obFichier.close()
```

- la méthode `write()` réalise l'écriture proprement dite. Les données à écrire doivent être fournies en argument.
- La méthode `close()` referme le fichier

34

Lecture séquentielle

- Il est possible de réouvrir le fichier créé, de manière à pouvoir y relire les informations enregistrées

```
#Lecture séquentielle  
ofi = open('Monfichier', 'r')  
t = ofi.read()  
print(t)  
Bonjour, fichier ! Quel beau temps, aujourd'hui !  
ofi.close()
```

Nom du fichier à ouvrir

referme le fichier après usage

- La méthode `read()` lit les données présentes dans le fichier et les transfère dans une variable de type chaîne de caractères (string)
- argument `'r'` = ouvrir en mode lecture

35

Fichier texte

- Lors des opérations de lecture, les lignes d'un fichier texte peuvent être extraites séparément les unes des autres.
- la méthode `readline()` ne lit qu'une seule ligne à la fois

```
#Fichier texte  
f = open("Fichier texte", "r")  
t = f.readline() #t contient la première ligne du fichier  
print(t) #Ligne 1  
print(f.readline()) #Ligne 2
```

Chaque `readline()` lit la ligne suivante dans le fichier

36

Fichier texte

- La méthode `readlines()` transfère toutes les lignes restantes dans une liste de chaînes

```
#Fichier texte
f = open("Fichier texte", "r")
t = f.readlines()
print(t)
print('Voici la ligne trois\n', t[2], 'Voici la ligne
quatre\n', t[3])
f.close()
```

Liste de chaînes

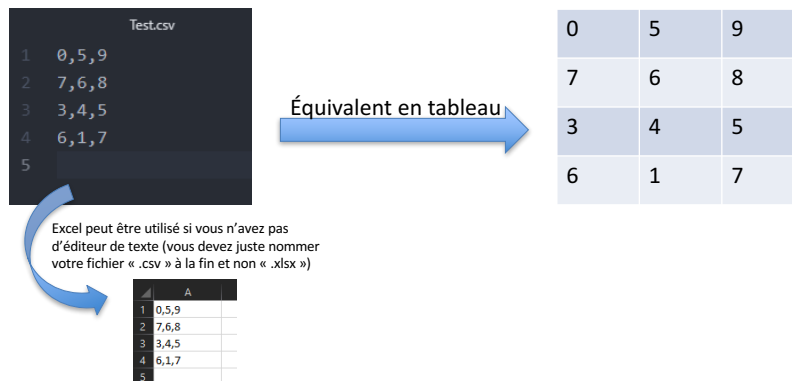
Le caractère '\n' signifie : à la ligne

37

Lecture/écriture d'un fichier « .csv »

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

- Un fichier .csv 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



39

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 :

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

40

Créer 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.
→ En effet, lors de la lecture d'un fichier, vous obtenez des strings.
- Cela peut vous paraître fastidieux mais si vous avez bien compris toute la matière antérieure, c'est un jeu d'enfant !

Raisonnement à suivre :

- 1) Ouvrir le fichier
- 2) Créer une variable qui est une liste (pour créer notre matrice d'integer)
- 3) Lire ligne par ligne le fichier, changer ces valeurs en integer, les mettre dans une liste que l'on va mettre dans notre liste
- 4) Fermer le fichier.
- 5) Nous avons notre matrice d'integer sur laquelle on peut réaliser des opérations

!/ Il faut toujours réaliser des codes les plus généraux possible ! Cela veut dire qu'il ne faut pas « hard coder » mais réaliser un code qui est indépendant du fichier .csv (qui fonctionne pour tous les fichiers .csv bien écrit) → Le code doit être correct peu importe le nombre de lignes, de colonnes, ... qu'a le fichier (en supposant que les données peuvent être changées en Int et que le fichier est bien écrit

41

Solution :

```
import csv

def get_a_list(fichier) :
    f = open(fichier, "r")
    fichier_ouvert = csv.reader(f)
    lst = []
    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

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

Le résultat avec un
fichier .csv qui contient :

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

Est :

```
[[9, 8, 2, 7], [6, 8, 7, 4], [3, 5, 9, 4], [8, 9, 0, 5], [9, 3, 7, 8]]
```

Nous avons bien une matrice d'integer !

!/ Remarquer 3 choses :

- 1) Nous avons utilisé plusieurs notions vues au cours :
 - Les fonctions (prébuild et original)
 - Les fichiers
 - La conversion de données (string en int)
 - Des variables
 - Des listes
 - ...
- 2) Ce code fonctionne pour tout autre fichier .csv bien écrit, nous n'avons pas « hard coder » pour qu'il fonctionne uniquement pour le fichier donné en exemple
- 3) Nous pouvons ajouter à ce code les try/except, donc des exceptions si le fichier donné en argument n'existe pas. Cela introduit la matière des dias suivantes

Subtilité

- Il est possible, dans le fichier .csv, que les chiffres ne soient pas séparés par une virgule mais pas un autre caractère.
- Dans ce cas, on peut décider où l'on veut que la séparation entre les valeurs se fasse
- Ceci se fait avec l'instruction `csv.reader(<fichier>, delimiter = '<symbole>')`

Exemple :

Si le fichier .csv est le suivant :

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

Donc, par défaut, la séparation des valeurs par python se fait sur les virgules, mais on peut changer cela en ajoutant `delimiter '<symbole>'`

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

Résultat :

```
['9#8#2#7']
['6#8#7#4']
['3#5#9#4']
['8#9#0#5']
['9#3#7#8']
```

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

Résultat :

```
['9', '8', '2', '7']
['6', '8', '7', '4']
['3', '5', '9', '4']
['8', '9', '0', '5']
['9', '3', '7', '8']
```

On change sur quel symbole doit être faite la séparation des valeurs

```
import csv
f = open("Test.csv", "r")
fichier = csv.reader(f, delimiter="8")
for line in fichier:
    print(line)
f.close()
```

Résultat :

```
['9#', '8#2#7']
['6#', '8#7#4']
['3#5#9#4']
['', '8#9#0#5']
['9#3#7#8', '']
```

43

écrire 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 :

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

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

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
```

44

Exceptions

Exceptions

- Opérations effectuées lorsqu'une erreur est détectée au cours de l'exécution d'un programme.
- 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

46

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.

47

Mécanisme de traitement des exceptions

- En Python, il est également possible d'associer un mécanisme de surveillance à tout un ensemble d'instructions.
- Simplifie le traitement des erreurs qui peuvent se produire dans cet ensemble d'instructions.
- On parle d'intercepter une erreur et d'alors exécuter des instructions spécifiques pour traiter cette erreur.
- Python: bloc **try – except – else**

48

Bloc try – except – else


- 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 ne s'est produite dans les instructions qui suivent **try**, 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.

49

Exemple

```
#Bloc try - except - else
def existe(fname):
    try:
        f = open(fname, 'r')
        f.close()
        return 1
    except:
        return 0

filename = input("Veuillez entrer le nom du fichier : ")
if existe(filename): #1 = True, 0 = False
    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

50

Application fichiers et exceptions

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)
```

52