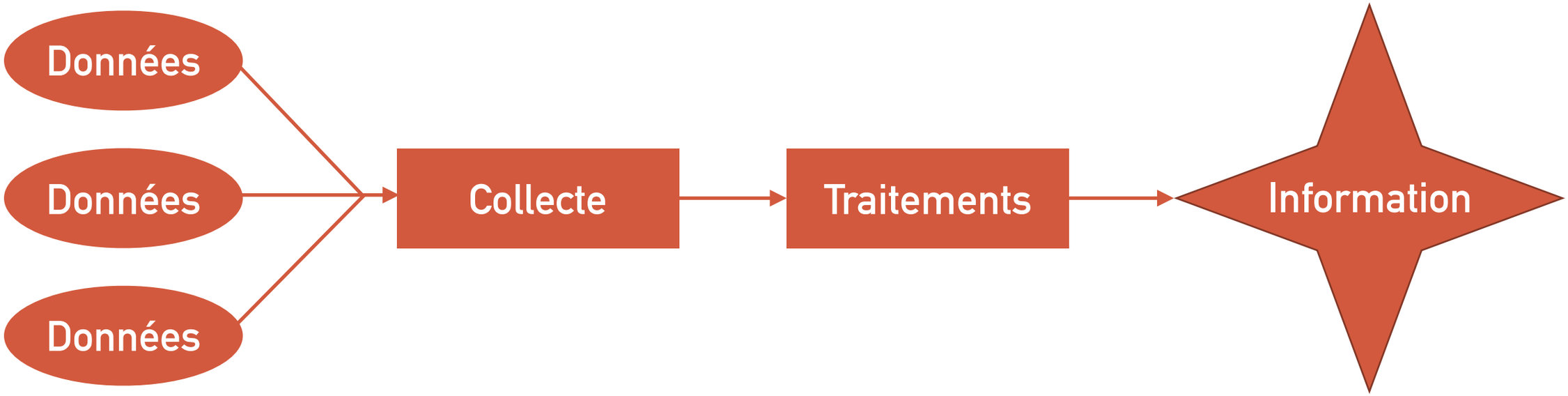
TRAITEMENT DE L'INFORMATION

Ophélie Fraisier
ophelie.fraisier@irit.fr

2018 — 2019

POURQUOI FAIRE ?

- Consolidation



- Sources très variées
  - Structurées (bases de données, XML, …)
  - Non structurées (pages web, fichiers de log, …)

DIFFÉRENTS FORMATS DE FICHIERS

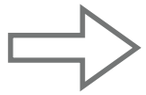
- Plusieurs formats ouverts de fichiers de données existants
- Possibilité de croiser des fichiers propriétaires
  - .rdata (R)
  - .sas7bdat (SAS)
  - .xls (Excel)

DIFFÉRENTS FORMATS DE FICHIERS : CSV

Comma-separated values

- Représentation textuelle d'un tableau
  - Une ligne du fichier = une ligne du tableau
  - Colonnes séparées par une virgule

Sexe	Prénom	Année de naissance
M	Alphonse	1932
F	Béatrice	1964
F	Charlotte	1988



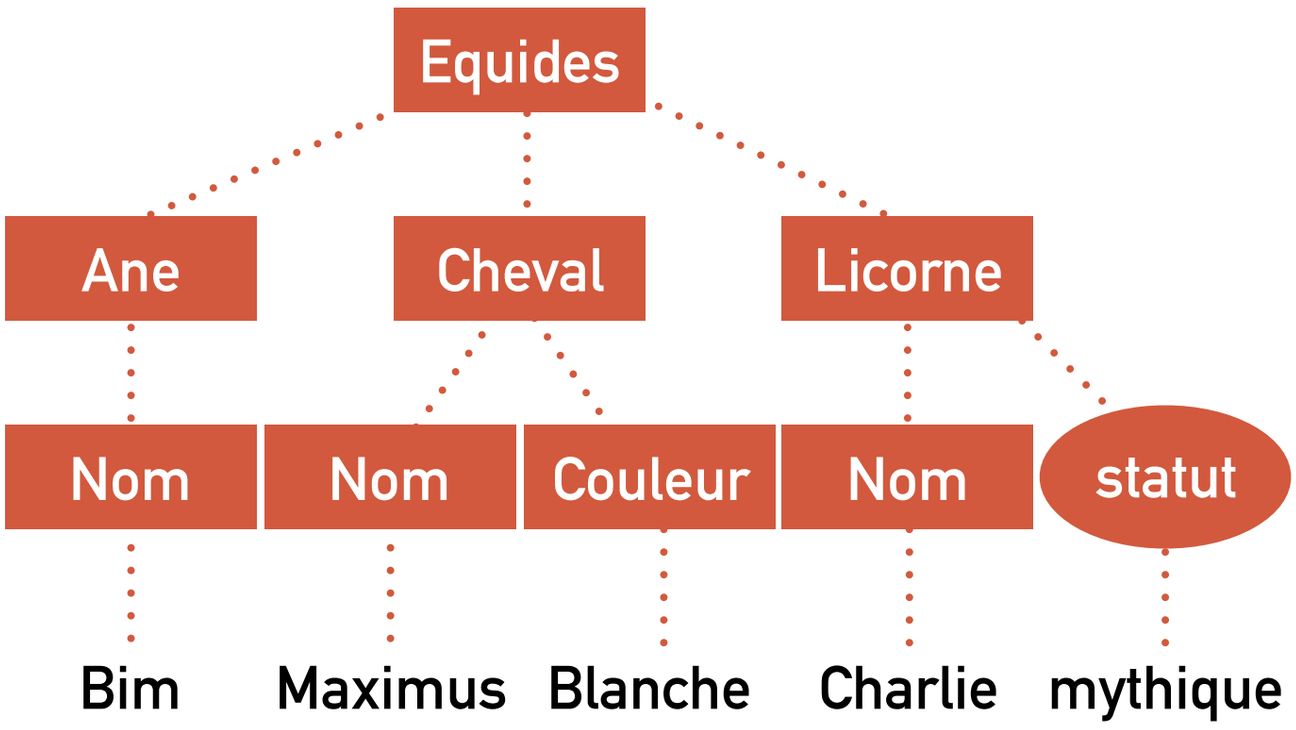
```
Sexe,Prénom,Année de naissance
M,Alphonse,1932
F,Béatrice,1964
F,Charlotte,1988
```

- Pas de spécification formelle donc variantes possibles
  - Virgules remplacées par des points-virgules
  - Champs texte entourés par des guillemets
  - …
- Cousin du CSV : le TSV
  - Tab-separated values : champs séparés par des tabulations

DIFFÉRENTS FORMATS DE FICHIERS : XML

eXtensible Markup Language

- Représentation textuelle d'un arbre
  - 1 racine unique
  - Éléments imbriqués délimités par `<balise></balise>`
  - Les noeuds sans enfants contiennent du texte
  - Possibilité de rajouter des attributs `<balise attribut="valeur">`




```
<equides>
   <ane>
      <nom>Bim</nom>
   </ane>
   <cheval>
      <nom>Maximus</nom>
      <couleur>Blanche</couleur>
   </cheval>
   <licorne statut="mythique">
      <nom>Charlie</nom>
   </licorne>
</equides>
```

DIFFÉRENTS FORMATS DE FICHIERS : JSON

JavaScript Object Notation

- 2 structures de base
  - les objets : `{clé1:valeur1, clé2:valeur2, …}`
  - les listes : `[valeur1, valeur2, …]`
- Une clé est une chaîne de caractères
- Une valeur peut être
  - une chaîne de caractères
  - un nombre
  - un objet
  - une liste
  - un booléen : `true` / `false`
  - une valeur nulle : `null`

```
{
    "menu": {
        "id": "file",
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

- Possibilité de combiner les structures

EXTRACTION DE DONNÉES NON STRUCTURÉES

- Il arrivera souvent que les données à récupérer ne soient pas accessibles sous forme structurée
  - Pages web
  - Fichiers de logs
  - Documents PDF
  - …

INTRODUCTION AUX EXPRESSIONS RÉGULIÈRES (REGEX)

- Motif décrivant un ensemble de chaînes de caractères possibles

```
[a-zA-Z-]+@[a-zA-Z-]+\.[a-zA-Z]{2,6}
```

Caractère entre a et z ou entre A et Z

1 fois ou plus

Le caractère « . »

Entre 2 et 6 fois

⇨ *Motif décrivant une adresse email simple*

- 2 comportements
  - « greedy » : correspondance avec le plus de caractères possibles
  - « lazy » : correspondance arrêtée le plus tôt possible, obtenu en ajoutant « ? » derrière le quantifieur

Lazy : `<.+?>`

```
<em>Hello</em>
```

Greedy : `<.+>`

- Site permettant de tester ses motifs : regex101.com

PENSE-BÊTE REGEX

Groupes et intervalles

`.`	N'importe quel caractère à part \n
`a|b`	a ou b
`[abc]`	a, b ou c
`[^abc]`	Tout sauf a, b ou c
`[a-z]`	Lettre entre a et z
`[A-Z]`	Lettre entre A et Z
`[0-9]`	Chiffre entre 0 et 9
`(…)`	Groupe nommé
`(mi|fa)`	« mi » ou « fa »

Ancres

`^`	Début de chaîne
`$`	Fin de chaîne
`\b`	Limite de mot
`\B`	Tout sauf une limite de mot

Classes de caractères

`\s`	Caractère espace
`\S`	Tout sauf un caractère espace
`\d`	Chiffre
`\D`	Tout sauf un chiffre
`\w`	Caractère alphanumérique
`\W`	Tout sauf un caractère alphanumérique

Quantifieurs

`*`	0 ou plus
`*?`	0+ (lazy)
`+`	1 ou plus
`+?`	1+ (lazy)
`?`	0 ou 1
`{3}`	Exactement 3
`{3,}`	3 ou plus
`{3,5}`	3, 4 ou 5

Caractères spéciaux

`\n`	Nouvelle ligne
`\r`	Retour chariot
`\t`	Tabulation
`\`	Caractère d'échappement

INTRODUCTION AU LANGAGE PYTHON

- Python est un langage de script très utilisé pour le traitement des données
- Orienté objet
- Multi-plateformes
- Sous licence libre

- À savoir :
  - Python3 n'est pas rétrocompatible avec Python2

PYTHON : PRINCIPES ET STRUCTURES DE BASE

- Blocs identifiés par l'indentation
- Types de base : `int`, `float`, `str`, `list`, `dict` (non exhaustif)
- Type non précisé lors de la déclaration des variables
- Listes :
  - Déclaration d'une liste vide : `liste = []`
  - Ajout d'un élément : `liste.append("Bonjour")`
  - Accès au $i^{ème}$ élément : `liste[i-1]` (l'index commence à 0)
- Dictionnaire :
  - Déclaration d'un dictionnaire vide : `dico = {}`
  - Ajout d'un élément : `dico["slt"] = "Bonjour"`
  - Accès à un élément : `dico["slt"]`
  - Accès toutes les clés / valeurs : `dico.keys()` / `dico.values()`
- Plusieurs types possibles dans la même liste / dictionnaire

PYTHON : PRINCIPES ET STRUCTURES DE BASE

- Condition :

```
if <condition>:
    <action>
elif <condition>:
    <action>
else:
    <action>
```

Optionnel

- Boucles :
  - Pour chaque

```
for <variable> in <liste>:
    <action>
```

  - Tant que

```
while <variable> <condition>:
    <action>
```

- Fonction :

```
def <nom_fonction>(<params>):
    <action>
    return <résultat>
```

Optionnel

PYTHON : PRINCIPES ET STRUCTURES DE BASE

```python
# Fonction permettant de saluer un nom
def greet(name):
    print("Hello {}".format(name)) # {} : remplacé par name


your_name = input("What is your name?\n")
greet(your_name)
friends = []
i = 0
while i < 5:
    friend_name = input("What are your fr
    if friend_name != "": # != veut dire
        friends.append(friend_name)
    i = i + 1
for f in friends:
    greet(f)
```

```
What is your name?
Ophélie
Hello Ophélie
What are your friends' names?
Un
What are your friends' names?
Deux
What are your friends' names?
Trois
What are your friends' names?
Quatre
What are your friends' names?
Cinq
Hello Un
Hello Deux
Hello Trois
Hello Quatre
Hello Cinq
```

PYTHON : PRINCIPES ET STRUCTURES DE BASE

```python
# Fonction testant le type
# de la variable
def traitement(x):
    if type(x) is int:
        return str(x**3)
    elif type(x) is str:
        return x.upper()
    else:
        return "Type inconnu"

# Utilisation d'un dictionnaire
dict = {"lundi": 10,
        "mardi": 20,
        30: "mercredi"}
for cle, valeur in dict.items():
    print("{} : {}".format(cle, traitement(valeur))
```

```python
# Opérations sur strings
# et listes
ch = "Licorne"
print(len(ch)) # 7
print(ch[0]) # "L"
print(ch[3]) # "o"

# [i:j] = de i à j-1
print(ch[1:3]) # "ic"
print(ch[4:]) # "rne"
print(ch[:5]) # "Licor"

# Casse d'une string
ch.lower() # licorne
ch.upper() # LICORNE
```

PYTHON : PRINCIPES ET STRUCTURES DE BASE

```
a = [0, 1, 2, 0, 1, 2, 3, 4, 5, 6]
b = [5, 6, 7, 8, 9, 10, 11]
# Enlever les doublons d'une liste
list(set(a)) # [0, 1, 2, 3, 4, 5, 6]
# Calculer l'intersection de 2 listes : & veut dire ET
list(set(a) & set(b)) # [5, 6]
# Calculer l'union de 2 listes : | veut dire OU
list(set(a) | set(b)) # [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
# Tester si un élément est dans la liste
7 in a # False
7 in b # True
# Même fonctionnement avec les chaînes de caractères
"Peter" in "Hello Peter" # True

# Compter les éléments d'une liste
import collections
count = collections.Counter(a) # créé un dictionnaire
count[1] # renvoie le nombre d'occurence de 1 dans a
count.most_common(1) # renvoie l'élément le plus présent
count.most_common() # renvoie les éléments par ordre décroissant
```

PYTHON : UTILISER LES REGEX

- Doc : https://docs.python.org/3.5/library/re.html

```
import re

phrase = "Cats are Satan's minions"
results = re.search("(.+) are (.+)", phrase)
if results is None:
    print("Pas de résultats")
else:
    print(results.group())  # toute la correspondance
    print(results.group(1)) # sous-groupe 1 : Cats
    print(results.group(2)) # sous-groupe 2 : Satan's minions
    # pour retourner tous les sous-groupes d'un coup
    print(results.groups()) # ["Cats", "Satan's minions"]
# Rechercher et remplacer
new_phrase = re.sub("Cats", "Dogs", phrase)
print(new_phrase) # Dogs are Satan's minions
```

PYTHON : LIRE ET ÉCRIRE DANS UN FICHIER

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists

```
# Lire un fichier
f1 = open("makers.txt", "r")
print(f1.read()) # lit l'intégralité du fichier
f1.close()
f2 = open("machines.txt", "r")
print(f2.readline()) # lit une ligne
for ligne in f2: # lit ligne par ligne jusqu'à la fin
    print(ligne)
f2.close()

# Écrire dans un fichier
f = open("makers.txt", "w")
f.write("New line")
f.close()
```

PYTHON : LIRE UN FICHIER CSV

- Parcours du fichier ligne par ligne
  - Arrivé à la fin, il faut ré-ouvrir le fichier si on souhaite le parcourir à nouveau
- Doc : https://docs.python.org/3.5/library/csv.html

```
fruit,qte
pomme,12
poire,7
banane,10
abricot,23
orange,2
```

```
     fruit  qte
0    pomme   12
3  abricot   23
```

```python
import sys
import csv

# Ouvre le fichier csv passé en argument au script
file_to_read = open(file_name, "r")
fcsv = csv.DictReader(file_to_read, delimiter=",")
# Sélection des lignes pour lesquels qte>10
for row in fcsv: # Row = dictionnaire
    if int(row["qte"]) > 10:
        print(row)
file_to_read.close()
```

PYTHON : LIRE UN FICHIER JSON

- Important de connaître la structure de son fichier JSON
- Doc : https://docs.python.org/3.5/library/json.html

```
[
    {"prénom": "Raiponce", "spécificité": "cheveux magiques"},
    {"nom": "Rider", "prénom": "Flynn"},
    {"nom": "Maximus", "spécificité": "cheval"}
]
```

```python
import sys
import json

# Ouvre le fichier json passé en argument
file_to_read = open(file_name, "r")
fjson = json.load(file_to_read)
# Parcours du fichier
for json_object in fjson:
    for key in json_object.keys():
        print("{} : {}".format(key, json_object[key]))
# Accès direct à une information
print(fjson[2]["nom"]) # Maximus
file_to_read.close()
```

```
prénom : Raiponce
spécificité : cheveux magiques
nom : Rider
prénom : Flynn
nom : Maximus
spécificité : cheval
```

[AVANCÉ] PYTHON : COMPRÉHENSIONS DE LISTE

- Syntaxe de boucles et de conditions plus compacte

```
a = []
for i in range(0, 10):
    if i % 2 == 0: # % est le modulo
        a.append(i)
```

```
a = [i for i in range(0, 10) if i % 2 == 0]
```

```
listOfWords = ["this","is","a","list","of","words"]
items = [word[0] for word in listOfWords]
```

```
d = {}
for n in range(0, 5):
    d[n] = n**2 # ** est la puissance
```

```
d = {n: n**2 for n in range(0, 5)}
```

[AVANCÉ] PYTHON : LIRE UN FICHIER CSV AVEC PANDAS

- Bibliothèque Python permettant de manipuler un fichier CSV
- Tableau de données Pandas = Dataframe
- Doc : http://pandas.pydata.org/pandas-docs/version/0.17.0/

```
import sys
import pandas

# Ouvre le fichier csv passé en argument au script
fcsv = pandas.read_csv(sys.argv[1], sep=",")
# Sélection des lignes pour lesquels qte>10
# Renvoie un dataframe
lignes = fcsv[ fcsv["qte"]>10 ]
# Affiche les colonnes "fruit" et "qte" du dataframe
print(lignes[["fruit", "qte"]])
# Ajout d'une colonne au dataframe
new_col = [i+25 for i in fcsv["qte"]]
fcsv["qte_max"] = new_col
print(fcsv)
# pandas ferme automatiquement le fichier
```