Ressource R208
Analyse et traitement de données structurées

IUT de Béziers, dépt. R&T © 2022-2024
http://www.borelly.net/
Christophe.BORELLY@umontpellier.fr

Contenus de la ressource

- Structure d’un programme : arborescence de fichiers, modules et packages.
- Contexte d’exécution : programme principal vs module
- Structure complexe de données :
  - Listes 2D, tableaux associatifs/dictionnaires
  - Notion de classes (instance, attributs, méthodes)
- Manipulation de fichiers avancée :
  - Fichiers structurés (XML, CSV, JSON, YAML)
  - Gestion de l’arborescence par le code
  - Lecture/écriture de fichiers structurés
  - Notion de sérialisation
  - Notion de persistance des données
- Initiation aux expressions régulières
- Introduction au traitement des erreurs

Généralités

Fichiers structurés (XML, CSV, JSON, YAML, ...)

- eXtensible Markup Language
- Comma Separated Values
- JavaScript Object Notation
- Yet Another Markup Language

Cours 2+1 (1h15), **TD** 3 (1h15), **TP** 3 (2h45)

XML

- Langage de balisage extensible
- Exemple :

```
<?xml version="1.0"?>
<gpx version="1.1" creator="gpxgenerator.com">
<wpt lat="43.347821302710365" lon="3.223332513792312">
    <ele unit="m">61.07</ele>
    <time>2021-11-19T13:52:21Z</time>
</wpt>
<wpt lat="43.347864213753404" lon="3.2223937406370995">
    <ele unit="m">62.97</ele>
    <time>2021-11-19T13:52:27Z</time>
</wpt>
</gpx>
```

Lecture de fichier XML

```
import xml.etree.ElementTree as ET
tree=ET.parse('trajet1.gpx')
root=tree.getroot()
print('Racine:',root.tag)
print('Attributs:',root.attrib)
for child in root:
  print(child.tag,'=>',child.attrib)
  ele=child.find('ele')
  print('  Alti:',ele.text,ele.get('unit'))
  print('  Date:',child.find('time').text)
```

Résultat de lecture XML

```
Racine: gpx
Attributs: {'version': '1.1', 'creator': 'gpxgenerator.com'}
wpt => {'lat': '43.347821302710365', 'lon': '3.223332513792312'}
  Alti: 61.07 m
  Date: 2021-11-19T13:52:21Z
wpt => {'lat': '43.347864213753404', 'lon': '3.2223937406370995'}
  Alti: 62.97 m
  Date: 2021-11-19T13:52:27Z
wpt => {'lat': '43.347505320459234', 'lon': '3.2213423147032616'}
  Alti: 63.85 m
  Date: 2021-11-19T13:52:35Z
wpt => {'lat': '43.346810933930804', 'lon': '3.221728552801406'}
  Alti: 61.46 m
  Date: 2021-11-19T13:52:42Z
wpt => {'lat': '43.3473367509018', 'lon': '3.2221630706618187'}
  Alti: 60.11 m
  Date: 2021-11-19T13:52:48Z
...
```

Utilisation de XPATH

- La librairie python **lxml** permet d'utiliser la syntaxe XPATH.
- Extrait de la syntaxe XPATH :
  - `.` : l'élément courant
  - `..` : l'élément parent
  - `/xxx/yyy` : les éléments yyy enfants de l'élément xxx (nom absolu)
  - `//yyy` : tous les éléments yyy du document
  - `/xxx/yyy[3]` : le 3ème élément yyy
  - `/xxx/yyy/@zz` : l'attribut zz de l'élément yyy
  - `/xxx/yyy[@zz='aa']` : l'élément yyy dont l'attribut zz est égal à « aa »
  - `/xxx/yyy[ww>10]`: l'élément yyy dont un enfant ww est supérieur à 10

Exemple XPATH

```
from lxml import etree
tree=etree.parse('trajet1.gpx')
n=tree.xpath('count(//wpt)')
print(f'Nombre de points GPS: {n}')
lat=tree.xpath('//wpt/@lat')
for p in lat:
  print(f' Latitude: {p}')
```

```
Nombre de points GPS: 6.0
 Latitude: 43.347821302710365
 Latitude: 43.34786421375340 4
 Latitude: 43.34750532045923 4
 Latitude: 43.34681093393080 4
 Latitude: 43.34733367509018
 Latitude: 43.34711334953765
```

Écriture de fichier XML

```python
import xml.etree.ElementTree as ET
matieres=['math','physique','français','math','sport']
notes=[14,12.2,17,13.4,15.5]
root=ET.Element('resultats')
root.set('prenom','John')
root.set('nom','DOE')
for i,n in enumerate(notes):
  note=ET.Element('note')
  note.text=str(n)
  note.set('matiere',matieres[i])
  root.append(note)
tree=ET.ElementTree(root)
ET.indent(tree,space='  ',level=0)
tree.write('notes.xml',encoding='utf-8',xml_declaration=True)
```

```
<?xml version='1.0' encoding='utf-8'?>
<resultats prenom="John" nom="DOE">
  <note matiere="math">14</note>
  <note matiere="physique">12.2</note>
  <note matiere="français">17</note>
  <note matiere="math">13.4</note>
  <note matiere="sport">15.5</note>
</resultats>
```

CSV

- Valeurs séparées par des virgules et « protégées » par des guillemets.
- RFC 4180
- Exemple :

```
Year,Make,Model,Comment
2020,Mazerrati,MC20,My car
1997,Ford,E350,"Super, luxurious truck"
1919,Citroën,Type A,"Very ""old"" car"
```

Lecture CSV (1)

```
import csv
with open('cars.csv') as csvfile:
    reader=csv.reader(csvfile)
    next(reader)# Skip first line
    for row in reader:
        print(row)
```

```
['2020', 'Mazerrati', 'MC20', 'My car']
['1997', 'Ford', 'E350', 'Super, luxurious truck']
['1919', 'Citroën', 'Type A', 'Very "old" car']
```

Lecture CSV (2)

```
import csv
with open('cars.csv') as csvfile:
    reader=csv.DictReader(csvfile)
    for row in reader:
        print(row)
```

```
{'Year':'2020', 'Make':'Mazerrati', 'Model':'MC20', 'Comment':'My car'}
{'Year':'1997', 'Make':'Ford', 'Model':'E350', 'Comment':'Super, luxurious truck'}
{'Year':'1919', 'Make':'Citroën', 'Model':'Type A', 'Comment':'Very "old" car'}
```

Écriture CSV

```
import csv
data={'A':['OM','PSG','LOSC','SR'],'B':['ASSE','RCS']}
with open('equipes.csv','w') as csvout:
  writer=csv.writer(csvout)
  writer.writerow(['groupe','equipe'])#header
  for grp,v in data.items():
    for equipe in v:
      writer.writerow([grp,equipe])
```

Voir aussi `csv.DictWriter()`...

```
groupe,equipe
A,OM
A,PSG
A,LOSC
A,SR
B,ASSE
B,RCS
```

JSON

- JavaScript Object Notation :
- Exemple :

```
{
    "menu": {
        "id": 1234,
        "value": "File",
        "popup": {
            "menuitem": [
                { "value": "New", "onclick": "CreateNewDoc()" },
                { "value": "Open", "onclick": "OpenDoc()" },
                { "value": "Close", "onclick": "CloseDoc()" }
            ]
        }
    }
}
```

Lecture de données JSON

```
import json
data=json.loads(open('menus.json').read())
print('ID :',data['menu']['id'])
print('Val:',data['menu']['value'])
for d in data['menu']['popup']['menuitem']:
  print('  ',d['value'])
```

```
ID : 1234
Val: File
    New
    Open
    Close
```

Écriture de données JSON

```
import json
r=json.dumps(['foo',{'bar':('baz',None,1.0,2)}])
print(r)
```

```
["foo", {"bar": ["baz", null, 1.0, 2]}]
```

YAML (1)

- Les commentaires sont signalés par le signe dièse (`#`) et se prolongent sur toute la ligne. Si par contre le dièse apparait dans une chaîne, il signifie alors un nombre littéral.
- Une valeur nulle s'écrit avec le caractère tilde (`~`)
- Il est possible d'inclure une syntaxe JSON dans une syntaxe YAML.
- Les éléments de listes sont dénotés par le tiret (`-`), suivi d'un espace, à raison d'un élément par ligne.
- Les tableaux sont de la forme `clé: valeur`, à raison d'un couple par ligne.

https://fr.wikipedia.org/wiki/YAML

YAML (2)

- Les scalaires peuvent être entourés de guillemets doubles (`"`), ou simples (`'`), sachant qu'un guillemet s'échappe avec un antislash (`\`), alors qu'une apostrophe s'échappe avec une autre apostrophe. Ils peuvent de plus être représentés par un bloc indenté avec des modificateurs facultatifs pour conserver (`|`) ou éliminer (`>`) les retours à la ligne.
- Plusieurs documents rassemblés dans un seul fichier sont séparés par trois traits d'union (`---`) ; trois points (`...`) optionnels marquent la fin d'un document dans un fichier.
- Les nœuds répétés sont initialement signalés par une esperluette (`&`) puis sont référencés avec un astérisque (`*`) ; JSON, un langage concurrent de YAML, est compatible avec la syntaxe de JavaScript mais ne supporte pas cette notion de référence.
- L'indentation, par des espaces, manifeste une arborescence.

https://fr.wikipedia.org/wiki/YAML

Exemple YAML

```yaml
---
receipt:     Oz-Ware Purchase Invoice
date:        2012-08-06
customer:
    given:   Dorothy
    family:  Gale

items:
    - part_no:   A4786
      descrip:   Water Bucket (Filled)
      price:     1.47
      quantity:  4

    - part_no:   E1628
      descrip:   High Heeled "Ruby" Slippers
      size:      8
      price:     100.27
      quantity:  1

bill-to:  &id001
    street: |
            123 Tornado Alley
            Suite 16
    city:   East Centerville
    state:  KS

ship-to:  *id001

specialDelivery:  >
    Follow the Yellow Brick
    Road to the Emerald City.
    Pay no attention to the
    man behind the curtain.
...
```

Lecture YAML

```
from yaml import load
from yaml import Loader

stream=open('bill.yaml').read()
data=load(stream,Loader=Loader)
print(data)# JSON
```

Résultat en JSON


```
{'receipt': 'Oz-Ware Purchase Invoice', 'date':
datetime.date(2012, 8, 6), 'customer': {'given': 'Dorothy',
'family': 'Gale'}, 'items': [{'part_no': 'A4786',
'descrip': 'Water Bucket (Filled)', 'price': 1.47,
'quantity': 4}, {'part_no': 'E1628', 'descrip': 'High
Heeled "Ruby" Slippers', 'size': 8, 'price': 100.27,
'quantity': 1}], 'bill-to': {'street': '123 Tornado
Alley\nSuite 16\n', 'city': 'East Centerville', 'state':
'KS'}, 'ship-to': {'street': '123 Tornado Alley\nSuite
16\n', 'city': 'East Centerville', 'state': 'KS'},
'specialDelivery': 'Follow the Yellow Brick Road to the
Emerald City. Pay no attention to the man behind the
curtain.\n'}
```

Écriture YAML

```
from yaml import dump
from yaml import Dumper

data={'utilisateurs':[f'User{i}'
        for i in range(5)]}
output=dump(data,Dumper=Dumper)
print(output)
```

```
utilisateurs:
- User0
- User1
- User2
- User3
- User4
```

Sérialisation/Désérialisation

- La sérialisation (serialization, marchalling) est le codage d'une information sous la forme d'une suite de données plus petites (e.g. en octets) pour sa sauvegarde (persistance) ou son transport sur le réseau.
- L'activité réciproque, visant à décoder cette suite pour créer une copie conforme de l'information d'origine, s'appelle la désérialisation (deserialization, unmarshalling).

Sérialisation d'objets (1)

```
class gpsPoint:
  def __init__(self,nom,lon,lat):
    self.x=lon
    self.y=lat
    self.name=nom
  @staticmethod
  def from_json(jsonStr):
    return gpsPoint(jsonStr['nom'],jsonStr['lon'],
                    jsonStr['lat'])
  def to_json(self):
    return {'_class':'gpsPoint',
    '_fields':{'nom':self.name,'lon':self.x,'lat':self.y}}
  def __str__(self):
    return f'GPS({self.name},lon={self.x},lat={self.y})'
  def __repr__(self):
    return f'Point({self.name},lon={self.x},lat={self.y})'
```

Sérialisation d'objets (2)

```
import json
from random import randint

def json_encoder(obj):
  if isinstance(obj,gpsPoint):
    return obj.to_json()

r=[gpsPoint(f'p{i}',randint(-180,180),randint(-90,90))
           for i in range(3)]
print('Obj:',r[0])
print('Lst:',r)
js=json.dumps(r,default=json_encoder)
print('JSON:',js)
for o in json.loads(js):
  p=gspPoint.from_json(o['_fields'])
  print('Obj:',p)
```

Sérialisation d’objets (3)

```
Obj: GPS(p0,lon=-86,lat=-58)
Lst: [Point(p0,lon=-86,lat=-58),
Point(p1,lon=-93,lat=-41),
Point(p2,lon=-171,lat=-17)]
JSON: [{"_class": "gpsPoint", "_fields": {"nom":
"p0", "lon": -86, "lat": -58}}, {"_class":
"gpsPoint", "_fields": {"nom": "p1", "lon": -93,
"lat": -41}}, {"_class": "gpsPoint", "_fields":
{"nom": "p2", "lon": -171, "lat": -17}}]
Obj: GPS(p0,lon=-86,lat=-58)
Obj: GPS(p1,lon=-93,lat=-41)
Obj: GPS(p2,lon=-171,lat=-17)
```

Références

- https://docs.python.org/3.10/
- https://www.w3schools.com/python/python_reference.asp