Introducció

El format XML s'ha introduït en tots els camps (inclosos la traducció) per diversos motius, però en podem destacar uns pocs:

- És un format estàndard i obert, amb el que es poden crear documents per a qualsevol tipus d'aplicació.
- Hi ha tot un seguit d'eines estàndard que permeten tractar arxius XML d'una manera ràpida i fàcil. En aquest sentit, Python també disposa de moltes llibreries i mòduls que permeten crear i llegir arxius XML d'una manera molt senzilla.

En aquesta unitat veurem alguns exemples senzills i el lector podrà crear-se les seves pròpies aplicacions modificant els exemples que exposem. També es pot trobar molta més informació sobre Python i XML en els següents enllaços:

- https://docs.python.org/3/library/xml.html
- https://docs.python.org/3/library/xml.etree.elementtree.html

Farem servir diverses tècniques per llegir i obtenir informació d'arxius XML:

- Funcions de cadenes. Ens serviran en les casos en què les cerques siguin senzilles i en les que l'arxiu XML guardi un format molt homogeni.
- Expressions regulars: ens permetran fer operacions senzilles sobre XMLs. En aplicacions reals només s'ha d'escollir aquesta opció quan vulguem obtenir una informació molt concreta dels arxius. Per a tractaments més complexos, és millor fer servir les altres opcions que trobareu en aquesta mateixa unitat.
- El paquet xmltodict que ens transforma un arxiu XML en un diccionari de Python.
- Ús de la llibreria xml.etree, que ens proporciona moltes funcions per al tractament d'arxius XML.

En tota aquesta unitat farem servir com a arxiu XML un que representa una base de dades de CDs de música. El podeu trobar en els arxius d'aquesta unitat. Aprofiteu per obrir-lo amb un editor de textos. Aquest arxiu el presentem en tres versions:

- catalog.xml: És la versió bàsica, si l'obriu veureu que presenta un format molt homogeni.
- catalog-mod.xml: És el mateix arxiu i conté la mateixa informació, però ara la informació de title presenta un salt de línia addicional.
- catalog-mod2.xml: És el mateix arxiu XML, però tota la informació està en una sola línia.

Per a cada programa que presenteu serà interessant observar si l'estratègia que hem fet servir funciona correctament per a les tres versions d'aquest arxiu.

D'aquesta unitat disposeu dels següents arxius:

- La unitat en PDF: 10-XML-cat.pdf
- Els programes i arxius necessaris: programes10-cat.zip

10.1. Tractament d'arxius XML amb funcions de cadena

Els arxius XML es poden tractar amb funcions estàndard de cadena com en el programa-10-1.py.

```
import codecs

entrada=codecs.open("catalog.xml","r",encoding="utf-8")
sortida=codecs.open("catalog.txt","w",encoding="utf-8")
artist=""
title=""
year=""
for linia in entrada:
    linia=linia.rstrip().lstrip()
    if linia.startswith("</cd>"):
        if not artist=="" and not title=="" and not year=="":
            cadena=artist+"\t"+title+"\t"+year
```

```
            print(cadena)
            sortida.write(cadena+"\n")
    if linia.startswith("<title>") and linia.endswith("</title>"):
        title=linia[7:-8]
    elif linia.startswith("<artist>") and linia.endswith("</artist>"):
        artist=linia[8:-9]
    elif linia.startswith("<year>") and linia.endswith("</year>"):
        year=linia[6:-7]
```

Si ens fixem, en aquest codi veurem que només es fan servir funcions de cadena estàndar com startswith() i endswith(), per exemple.

Si l'executem, veurem que funciona correctament i que a la sortida ens dóna la informació d'artista, títol i any.

```
    Bob Dylan  Empire Burlesque 1985
Bonnie Tyler Hide your heart 1988
Dolly Parton Greatest Hits 1982
Gary Moore Still got the blues 1990
```

Aquesta estratègia pot ser vàlida, però es basa massa en la disposició física de les marques i la informació en el document. Proveu ara si aquest programa funciona bé per a catalog-mod.txt i catalog-mod2.txt.

10.2. Tractament d'arxius XML mitjançant expressions regulars

Per evitar el problema de la dependència de la disposició de la informació, podem fer ús d'expressions regulars, com en el següent programa (programa-10-2.py)

```
    import codecs
import re
entrada=codecs.open("catalog.xml","r",encoding="utf-8")
sortida=codecs.open("catalog.txt","w",encoding="utf-8")
artist=""
title=""
year=""
for linia in entrada:
    linia=linia.rstrip().lstrip()
    if linia.startswith("</cd>"):
        if not artist=="" and not title=="" and not year=="":
            cadena=artist+"\t"+title+"\t"+year
            print(cadena)
            sortida.write(cadena+"\n")
    else:
        m_title = re.search('<title>(.+?)</title>', linia)
        if m_title:
            title = m_title.group(1)
        m_artist = re.search('<artist>(.+?)</artist>', linia)
        if m_artist:
            artist = m_artist.group(1)
        m_year = re.search('<year>(.+?)</year>', linia)
        if m_year:
            year = m_year.group(1)
```

Proveu d'executar el programa amb els arxius catalog modificats. Els pot tractar tots?

10.3. Tractament d'arxius XML amb xmltodict

xml2dict ens permet tractar arxius XML d'una manera molt fàcil, ja que converteix l'arxiu XML en una estructura de dades de tipus diccionari.

En el programa-10-3.py podem observar com fer servir aquesta llibreria.

```
import xmltodict

xml=open('catalog.xml')
xmldict = xmltodict.parse(xml.read())


for cd in xmldict["catalog"]["cd"]:
 print(cd["artist"],cd["title"],cd["year"])
```

Proveu amb totes les modificacions de l'arxiu catalog i observeu si és capaç de processar-lo correctament.

10.4. Tractament d'arxius XML amb xml.etree

Disposem d'una sèrie de llibreries que ens faciliten molt la lectura d'arxius XML. Una d'elles és xml.etree.ElementTree. Per observar com funciona, executarem el programa-10-4.py i observarem la sortida:

```
import xml.etree.ElementTree as etree

for event, elem in etree.iterparse("catalog.xml",events=("start", "end")):
 print(event,elem,elem.tag,elem.attrib)
```

La llibreria és capaç de detectar quan hi ha un event (i hem seleccionat el principi (start) i el final (end)), el element afectat, l'etiqueta de l'element i l'atribut de l'element. En aquest programa de prova simplement escrivim aquesta informació:

```
start <Element 'catalog' at 0x7f456b5c3728> catalog {}
start <Element 'cd' at 0x7f4569d999f8> cd {'id': '1'}
start <Element 'title' at 0x7f4569d3c2c8> title {}
end <Element 'title' at 0x7f4569d3c2c8> title {}
start <Element 'artist' at 0x7f4569d3c318> artist {}
end <Element 'artist' at 0x7f4569d3c318> artist {}
```

Amb aquesta informació, podem fer un programa que llegeixi l'arxiu (programa-10.5.py):

```
import xml.etree.ElementTree as etree
import codecs
artist=""
title=""
year=""
sortida=codecs.open("catalog.txt","w",encoding="utf-8")
for event, elem in etree.iterparse("catalog.xml",events=("start", "end")):
    if event=="end" and elem.tag=="cd":
        cadena=artist+"\t"+title+"\t"+year
        print(cadena)
        sortida.write(cadena+"\n")
        artist=""
        title=""
        year=""
    if event=="end" and elem.tag=="title":
        title="".join(elem.itertext()).lstrip().rstrip()
    if event=="end" and elem.tag=="artist":
        artist="".join(elem.itertext()).lstrip().rstrip()
    if event=="end" and elem.tag=="year":
```

year="".join(elem.itertext()).lstrip().rstrip()

Comproveu que aquest programa funciona també bé amb el catalog modificat.

10.5. Tractament d'arxius TMX

En aquest apartat expliquem el desenvolupament d'un programa que transforma arxius TMC en text separat per tabulador. En aquesta unitat hem vist diverses estratègies per al tractament d'arxius XML. Habitualment farem servir alguna llibreria especialitzada en el tractament d'arxius XML, com per exemple la librería xml.etree.ElementTree, que ja coneixem d'aquesta unitat.

Treballarem amb un arxiu XML d'exemple: en-es.tmx (que es correspon al corpus ECB anglès-espanyol).

El programa complet es pot descarregar d'aquest enllaç: TMX2tabtxt.py

10.5.a. TMX

Com ja sabem, el format TMX (Translation Memory eXchange) és un format estàndard basat en XML per a l'intercanvi de memòries de traducció. A continuació podem observer un fragment:

```
<body>
  <tu>
   <tuv xml:lang="en"><seg>The European Central Bank</seg></tuv>
   <tuv xml:lang="es"><seg>Banco Central Europeo</seg></tuv>
  </tu>
...
```

Volem fer un programa que a partir d'un arxiu TMX, el codi de llengua 1 i el codi de llengua 2 creï un arxiu separat per tabuladors que contingui segment l1 tabulador segment l2. El nostre programa haurà de parsejar l'XML i recórrer els elements tu:

```
tree = ET.parse(fentrada)
root = tree.getroot()
for tu in root.iter('tu'):
```

De cada tu haurà de recórrer totes les tuv

```
for tuv in tu.iter('tuv'):
```

mirar el codi de llengua:

```
lang=tuv.attrib['{http://www.w3.org/XML/1998/namespace}lang']
```

agafar el text del seg

```
text=seg.text
```

y mirar si correspon a la llengua 1 o a la llengua 2:

```
if lang==l1: sl_text=text
elif lang==l2: tl_text=text
```

i per últim escriure a l'arxiu (i també mostrar per pantalla) els textos corresponents a la llengua 1 i a la llengua 2, comprovant abans que hi hagi text per a les dues llengües:

```
if not sl_text=="" and not tl_text=="":
```

```
        cadena=sl_text+"\t"+tl_text
        print(cadena)
        sortida.write(cadena+"\n")
```

Fixa't que per a cada cop que llegim una tu posem a "" (el forcem a que estiguin buits) els textos de la llengua 1 i de la llengua 2.

10.5.b. Codi complet del programa

```
import xml.etree.ElementTree as ET
import sys
import codecs

fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]

minrel=3
nmax=5
sortida=codecs.open(fsortida,"w",encoding="utf-8")

tree = ET.parse(fentrada)
root = tree.getroot()

for tu in root.iter('tu'):
  sl_text=""
  tl_text=""
  for tuv in tu.iter('tuv'):
     lang=tuv.attrib['{http://www.w3.org/XML/1998/namespace}lang']
     for seg in tuv.iter('seg'):
        text=seg.text
        if lang==l1: sl_text=text
        elif lang==l2: tl_text=text



  if not sl_text=="" and not tl_text=="":
     cadena=sl_text+"\t"+tl_text
     print(cadena)
     sortida.write(cadena+"\n")
```

Fixa't que el programa espera que donem com a paràmetres d'entrada l'arxiu TMX a transformar i els codis de llengua:

```
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
```

per executar el programa haurem de fer:

```
python3 TMX2tabtxt.py en-es.tmx memo-en-es.txt en es
```

i en l'arxiu memo-en-es.txt tindrem la memòria de traducció convertida a text tabulat.

Mitjançant la llibreria sys podem obtenir els paràmetres d'entrada del programa de manera molt senzilla. A la llista sys.argv tenim els paràmetres, però és molt important tenir en compte que a sys.argv[0] tenim el nom del programa i que el primer paràmetre està a sys.argv[1].

10.5.c. Escriptura d'arxius TMX

A l'apartat anterior hem vist com parsejar un TMX i crear un arxiu de text tabulat. Ara aprendrem a escriure arxius TMX fent un programa que passi d'un arxiu de text tabulat a un arxiu TMX.

Presentarem dues versions del programa. La primera versió simplement farà servir funcions de cadena per escriure-les a l'arxiu resultant. El programa és el tabtxt2TMXa.py:

```
import codecs
from xml.sax.saxutils import escape
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
entrada=codecs.open(fentrada,"r",encoding="utf-8")
sortida=codecs.open(fsortida,"w",encoding="utf-8")
cadena='<?xml version="1.0" encoding="UTF-8" ?>'
sortida.write(cadena+"\n")
cadena='<tmx version="1.4">'
sortida.write(cadena+"\n")
cadena='<header/>'
sortida.write(cadena+"\n")
cadena='  <body>'
sortida.write(cadena+"\n")
for linia in entrada:
  linia=linia.rstrip()
  camps=linia.split("\t")
  segment1=camps[0]
```

```
    segment2=camps[1]
    cadena='   <tu>'
    sortida.write(cadena+"\n")
    cadena='      <tuv xml:lang="'+l1+'"><seg>'+escape(segment1)+'</seg></tuv>'
    sortida.write(cadena+"\n")
    cadena='      <tuv xml:lang="'+l2+'"><seg>'+escape(segment2)+'</seg></tuv>'
    sortida.write(cadena+"\n")
    cadena='    </tu>'
    sortida.write(cadena+"\n")


cadena='  </body>'
sortida.write(cadena+"\n")
cadena='</tmx>'
sortida.write(cadena+"\n")
```

Com podeu veure, anem creant les cadenes necessàries concatenant els elements i les escrivim a l'arxiu amb write. Aquesta és una opció molt senzilla i eficient.

També és possible fer servir una llibreria específica per a escriure arxius XML, com en el programa tabtxt2TMXb.py. Com veurem, simplement va construint els elements amb els seus fills, etc i després ho escriu tot a disc. La funció prettify fa que la sortida tingui un bon aspecte visual:

```
import sys
import codecs
from xml.etree.ElementTree import Element, SubElement, Comment, tostring
from xml.etree import ElementTree
from xml.dom import minidom
def prettify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent="  ")
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
entrada=codecs.open(fentrada,"r",encoding="utf-8")
sortida=codecs.open(fsortida,"w",encoding="utf-8")
# Configure one attribute with set()
```

```
root = Element('tmx')
root.set('version', '1.4')
header = SubElement(root, 'header')
body = SubElement(root, 'body')
for linia in entrada:
  linia=linia.rstrip()
  camps=linia.split("\t")
  segment1=camps[0]
  segment2=camps[1]
  tu=SubElement(body, 'tu')
  tuv = SubElement(tu,'tuv')
  tuv.set('xml:lang',l1)
  seg= SubElement(tuv,'seg')
  seg.text=segment1

  tuv = SubElement(tu,'tuv')
  tuv.set('xml:lang',l2)
  seg= SubElement(tuv,'seg')
  seg.text=segment2
sortida.write((prettify(root)))
```

L'inconvenient d'aquesta estratègia és que anem ocupant memòria del sistema fins que guardem l'arxiu a disc i això per a arxius molt grans pot donar problemes. L'avantatge és que ens assegurem que el XML de sortida estarà ben format.

10.6. Tractament d'arxiusTBX

En aquest apartat aprendrem a tractar arxius TBX. Ho farem d'una manera molt similar als arxius TMX. Per poder-los tractar, necessitarem conèixer una mica com són aquests arxius. Teniu dispible l'arxiu ejemplo.tbx (mostrarem només un fragment):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE martif SYSTEM "TBXcoreStructV02.dtd">
<martif type="TBX-Default" xml:lang="en">
 <martifHeader>
  <fileDesc>
   <sourceDesc>
    <p>This is a TBX file downloaded from the IATE website. Address any enquiries to iate@cdt.europa.eu.</
p>
   </sourceDesc>
```

```
  </fileDesc>
  <encodingDesc>
    <p type="XCSURI">TBXXCS.xcs</p>
  </encodingDesc>
</martifHeader>
<text>
  <body>
    <termEntry id="IATE-28960">
      <descripGrp>
        <descrip type="subjectField">2826, 7611</descrip>
      </descripGrp>
      <langSet xml:lang="en">
        <tig>
          <term>EBU</term>
          <termNote type="termType">abbreviation</termNote>
          <descrip type="reliabilityCode">3</descrip>
        </tig>
        <tig>
          <term>European Blind Union</term>
          <termNote type="termType">fullForm</termNote>
          <descrip type="reliabilityCode">3</descrip>
        </tig>
      </langSet>
      <langSet xml:lang="es">
        <tig>
          <term>UEC</term>
          <termNote type="termType">abbreviation</termNote>
          <descrip type="reliabilityCode">3</descrip>
        </tig>
        <tig>
          <term>Unión Europea de Ciegos</term>
          <termNote type="termType">fullForm</termNote>
          <descrip type="reliabilityCode">3</descrip>
        </tig>
      </langSet>
      <langSet xml:lang="fr">
```

```
        <tig>
          <term>UEA</term>
          <termNote type="termType">abbreviation</termNote>
          <descrip type="reliabilityCode">2</descrip>
        </tig>
        <tig>
          <term>Union européenne des aveugles</term>
          <termNote type="termType">fullForm</termNote>
          <descrip type="reliabilityCode">3</descrip>
        </tig>
      </langSet>
    </termEntry>
....
```

En el primer programa (TBX2tabtxt.py) convertirem l'arxiu TBX en text tabulat. Fixa't en com demanem els paràmetres d'entrada:

```
import xml.etree.ElementTree as ET
import sys
import codecs
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
minrel=3
nmax=5
sortida=codecs.open(fsortida,"w",encoding="utf-8")
tree = ET.parse(fentrada)
root = tree.getroot()
for termEntry in root.iter('termEntry'):
    termL1=""
    termL2=""
    for langset in termEntry.iter('langSet'):
        lang=langset.attrib['{http://www.w3.org/XML/1998/namespace}lang']
        for tig in langset.iter('tig'):
            for term in tig.iter('term'):
                termino=term.text
                if lang==l1:
```

```
                termL1=termino
            elif lang==l2:
                termL2=termino
        if not termL1=="" and not termL2=="":
            cadena=termL1+"\t"+termL2
            print(cadena)
            sortida.write(cadena+"\n")
```

Per executar el programa hem d'escriure:

```
python3 TBX2tabtxt.py ejemplo.tbx ejemplo.txt en es
```

Hem introduit una modificació en el programa TBX2tabtxt2.py per a que a la sortida ens proporcioni l'ID i les àrees temàtiques:

```
import xml.etree.ElementTree as ET
import sys
import codecs
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
minrel=3
nmax=5
sortida=codecs.open(fsortida,"w",encoding="utf-8")
tree = ET.parse(fentrada)
root = tree.getroot()
for termEntry in root.iter('termEntry'):
    termL1=""
    termL2=""
    ident=termEntry.attrib['id']
    subjects=""
    for group in termEntry.iter('descripGrp'):

        for descrip in group.iter('descrip'):
            if descrip.attrib['type']=='subjectField':
                subjects=descrip.text
    for langset in termEntry.iter('langSet'):
        lang=langset.attrib['{http://www.w3.org/XML/1998/namespace}lang']
```

```
        for tig in langset.iter('tig'):
            for term in tig.iter('term'):
                termino=term.text
                if lang==l1:
                    termL1=termino
                elif lang==l2:
                    termL2=termino
        if not termL1=="" and not termL2=="":
            cadena=ident+"\t"+subjects+"\t"+termL1+"\t"+termL2
            print(cadena)
            sortida.write(cadena+"\n")
```

I una modificació més per a que només s'escrigui el terme si el reliabilityCode és 3 o superior:

```
import xml.etree.ElementTree as ET
import sys
import codecs
fentrada=sys.argv[1]
fsortida=sys.argv[2]
l1=sys.argv[3]
l2=sys.argv[4]
minrel=3
nmax=5
sortida=codecs.open(fsortida,"w",encoding="utf-8")
tree = ET.parse(fentrada)
root = tree.getroot()
for termEntry in root.iter('termEntry'):
    termL1=""
    termL2=""
    ident=termEntry.attrib['id']
    subjects=""
    reliability=0
    for group in termEntry.iter('descripGrp'):

        for descrip in group.iter('descrip'):
            if descrip.attrib['type']=='subjectField':
                subjects=descrip.text
```

```
for langset in termEntry.iter('langSet'):
    lang=langset.attrib['{http://www.w3.org/XML/1998/namespace}lang']
    for tig in langset.iter('tig'):
        for term in tig.iter('term'):
            termino=term.text
            if lang==l1:
                termL1=termino
            elif lang==l2:
                termL2=termino
        for descrip in tig.iter('descrip'):
            if descrip.attrib['type']=='reliabilityCode':
                reliability=int(descrip.text)



    if not termL1=="" and not termL2=="" and reliability>=3:
        cadena=ident+"\t"+subjects+"\t"+termL1+"\t"+termL2
        print(cadena)
        sortida.write(cadena+"\n")
```