Download notebook

XML

Simple example from http://software-carpentry.org/3_0/xml.html

```
<?xml version="1.0" encoding="utf-8"?>
<planet name="Mercury">
  <period units="days">87.97</period>
</planet>
```

XML files are best viewed as trees

Very common structured data format, supporting hierarchical, nested data with metadata.

Recommended libraries:

- `lxml` (Performance)
- `untangle` (Simplicity)

```
$ pip install untangle
```

untangle

Example data is an XML file with information about outages on escalators and elevators in the New York subway system

```
In [1]: xml_file = "nyct_ene.xml"
```

```
In [2]: import untangle
```

```
In [3]: doc = untangle.parse(xml_file)
```

```
In [4]: doc.get_elements()
```

```
Out[4]: [Element(name = NYCOutages, attributes = {}, cdata = )]
```

```
In [5]: doc.
```

```
  File "<ipython-input-5-baa82b978199>", line 1
    doc.
        ^
SyntaxError: invalid syntax
```

```
In [6]: outages = doc.NYCOutages.outage
```

```
In [7]: len(outages)
```

```
Out[7]: 38
```

```
In [8]: outage = outages[5]
```

```
In [9]: outage.get_elements()
```

```
Out[9]: [Element(name = station, attributes = {}, cdata = 181 ST STATION),
         Element(name = borough, attributes = {}, cdata = MN),
         Element(name = trainno, attributes = {}, cdata = 1),
         Element(name = equipment, attributes = {}, cdata = EL110),
         Element(name = equipmenttype, attributes = {}, cdata = EL),
         Element(name = serving, attributes = {}, cdata = LOWER MEZZANINE TO UPPER MEZZANINE),
         Element(name = ADA, attributes = {}, cdata = N),
         Element(name = outagedate, attributes = {}, cdata = 12/27/2012  8:07:00 AM),
         Element(name = estimatedreturntoservice, attributes = {}, cdata = 12/29/2012 12:00:00 AM),
         Element(name = reason, attributes = {}, cdata = REPAIR),
         Element(name = isupcomingoutage, attributes = {}, cdata = N),
         Element(name = ismaintenanceoutage, attributes = {}, cdata = N)]
```

```
In [10]: outage.estimatedreturntoservice.cdata
```

```
Out[10]: u'12/29/2012 12:00:00 AM'
```

CSV

Means a file where data just have some sort of separator

Example, small sample of http://a841-dotweb01.nyc.gov/datafeeds/ParkingReg/signs.CSV (88 MB)

```
In [11]: %%bash
         head thousand_signs.csv
```

```
B,P-004958,1,0000 ,   ,Curb Line
B,P-004958,2,0009 ,   ,Property Line
B,P-004958,3,0030 ,   ,NIGHT REGULATION (MOON & STARS SYMBOLS) NO PARKING (SANITATION BROOM SYMBOL) MIDNIGHT TO 3AM TUES
& FRI <--> (SUPERSEDED BY SP-841C)
B,P-004958,4,0030 ,   ,1 HOUR PARKING 9AM-7PM EXCEPT SUNDAY
B,P-004958,5,0208 ,   ,NIGHT REGULATION (MOON & STARS SYMBOLS) NO PARKING (SANITATION BROOM SYMBOL) MIDNIGHT TO 3AM TUES
& FRI <--> (SUPERSEDED BY SP-841C)
B,P-004958,6,0208 ,   ,1 HOUR PARKING 9AM-7PM EXCEPT SUNDAY
B,P-004958,7,0218 ,   ,Property Line
B,P-004958,8,0232 ,   ,Curb Line
```

```
B,P-009318,1,0000 ,   ,Curb Line
B,P-009318,2,0014 ,   ,Building Line
```

Builtin module `csv`

Primary documentation: http://docs.python.org/2.7/library/csv.html

In [12]:
```
# Simple reading

f = open("thousand_signs.csv", "r")
for _ in range(3):
    print(repr(f.readline()))
```

```
'B,P-004958,1,0000 ,   ,Curb Line\r\n'
'B,P-004958,2,0009 ,   ,Property Line\r\n'
'B,P-004958,3,0030 ,   ,NIGHT REGULATION (MOON & STARS SYMBOLS) NO PARKING (SANITATION BROOM SYMBOL) MIDNIGHT TO 3AM TUES
& FRI <--> (SUPERSEDED BY SP-841C)                                     \r\n'
```

In [13]:
```
import csv
```

In [14]:
```
# Using a CSV reader

f.seek(0)

reader = csv.reader(f)
reader
```

Out[14]:
```
<_csv.reader at 0x1084d54b0>
```

In [15]:
```
for _ in range(3):
    print(repr(reader.next()))
```

```
['B', 'P-004958', '1', '0000 ', '   ', 'Curb Line']
['B', 'P-004958', '2', '0009 ', '   ', 'Property Line']
['B', 'P-004958', '3', '0030 ', '   ', 'NIGHT REGULATION (MOON & STARS SYMBOLS) NO PARKING (SANITATION BROOM SYMBOL)
MIDNIGHT TO 3AM TUES & FRI <--> (SUPERSEDED BY SP-841C)                                     ']
```

In [16]:
```
f.close()
```

Rows are split in to lists based on the seperator on the fly

Dialects

There's no clear standard for the which characters are used for separation of columns and rows.

The specific setting of formatting is called a **dialect**

The default in `csv` is to use `,` for columns and `\r\n` for rows.

In [17]:
```
csv.list_dialects()
```

Out[17]:
```
['excel-tab', 'excel']
```

`excel-tab` uses `\t` for column separations

If we want to separate fields which internally uses `,`, we could for example use `;` as a separator.

In [18]:
```
%%bash
head example.csv
```

```
chapter; pages with footnotes; pages with references
1; 5,6; 1,2,4,5
2; 8; 7,8,9
3; 11,12; 10,12,13,14
4; 16,19; 16,17,20
```

In [19]:
```
csv.register_dialect("semicolon", delimiter=";", skipinitialspace=True)
csv.list_dialects()
```

Out[19]:
```
['excel-tab', 'excel', 'semicolon']
```

In [20]:
```
f = open("example.csv", "r")

reader = csv.reader(f, "semicolon")

for line in reader:
    print(line)

f.close()
```

```
['chapter', 'pages with footnotes', 'pages with references']
['1', '5,6', '1,2,4,5']
['2', '8', '7,8,9']
['3', '11,12', '10,12,13,14']
['4', '16,19', '16,17,20']
```

Automatically detecting dialect

In [21]:
```
sniffer = csv.Sniffer()
```

`csv.Sniffer` analyzes text for patterns, to find the dialect

In [22]:
```
f = open("example.csv", "r")
```

In [23]:
```
sample = f.read()

dialect = sniffer.sniff(sample)

f.seek(0)
reader = csv.reader(f, dialect)
for line in reader:
    print(line)

f.seek(0)
```

```
['chapter', 'pages with footnotes', 'pages with references']
['1', '5,6', '1,2,4,5']
['2', '8', '7,8,9']
['3', '11,12', '10,12,13,14']
['4', '16,19', '16,17,20']
```

`csv` can be very large, reading all of it to find the dialect can be time consuming and not necessary

In [24]:
```
sample = f.read(7)  # Read 7 bytes

print(repr(sample))
print("")

f.seek(0)

dialect = sniffer.sniff(sample)

reader = csv.reader(f, dialect)
for line in reader:
    print(line)

f.seek(0)
```

```
'chapter'

['chap', 'er; pages wi', 'h foo', 'no', 'es; pages wi', 'h references']
['1; 5,6; 1,2,4,5']
['2; 8; 7,8,9']
['3; 11,12; 10,12,13,14']
['4; 16,19; 16,17,20']
```

Undersampling can of course yield erronous dialects

In [25]:
```
sample = f.read(80)

print(repr(sample))
print("")

f.seek(0)

dialect = sniffer.sniff(sample)

reader = csv.reader(f, dialect)
for line in reader:
    print(line)

f.seek(0)
```

```
'chapter; pages with footnotes; pages with references\n1; 5,6; 1,2,4,5\n2; 8; 7,8,9'

['chapter', 'pages with footnotes', 'pages with references']
['1', '5,6', '1,2,4,5']
['2', '8', '7,8,9']
['3', '11,12', '10,12,13,14']
['4', '16,19', '16,17,20']
```

For larger files, around 1024 bytes is quite good, you want to have a few lines in the samples

In [26]:
```
dialect.delimiter, dialect.lineterminator, dialect.skipinitialspace
```

Out[26]:
```
(';', '\r\n', True)
```

In [27]:
```
sniffer.has_header(sample)
```

Out[27]:
```
True
```

In [28]:
```
f.close()
```

In [29]:
```
f = open("thousand_signs.csv", "r")
sample = f.read(100)
f.close

sniffer.has_header(sample)
```

Out[29]:
```
False
```

JSON

Builtin module `json`

JavaScript Object Notation, human readable serialization of data.

Mostly known from sending data between server and client in web applications

In Python nomenclature, one can say it consists of lists and dictionaries, which can be populated by strings, integers and floats

In [30]:
```
%%bash
# Example from wikipeda page on JSON
cat example.json
```

```
{
    "firstName": "John",
    "lastName": "Smith",
    "age": 25,
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": "10021"
    },
    "phoneNumber": [
        {
            "type": "home",
            "number": "212 555-1234"
        },
        {
            "type": "fax",
            "number": "646 555-4567"
        }
    ]
}
```

In [31]:
```
import json
```

In [32]:
```
f = open("example.json", "r")

json_data = json.load(f)

f.close()

json_data
```

Out[32]:
```
{u'address': {u'city': u'New York',
  u'postalCode': u'10021',
  u'state': u'NY',
  u'streetAddress': u'21 2nd Street'},
 u'age': 25,
 u'firstName': u'John',
 u'lastName': u'Smith',
 u'phoneNumber': [{u'number': u'212 555-1234', u'type': u'home'},
  {u'number': u'646 555-4567', u'type': u'fax'}]}
```

YAML

Package `pyyaml`, install by `pip install pyyaml`

Databases

Drivers for popular none-SQL databases:

- pycouchdb
- pymongo

Web services

A common way to make data available is in the form of a web service.

To get data one sends a `GET` command to a server, which interprets the `URI` used to send the `GET` command, and sends back what one asked for.

Python comes with several http libraries, but the recommendation is to install the package `requests`

(`pip install requests`)

Documentation on http://docs.python-requests.org/

In [33]:
```
import requests
```

In [34]:
```
r = requests.get("http://ws.spotify.com/search/1/track.json", params={"q": "kaizers orchestra"})
```

In [35]:
```
r.url
```

Out[35]:
```
u'http://ws.spotify.com/search/1/track.json?q=kaizers+orchestra'
```

In [36]:
```
r.status_code
```

Out[36]:
```
200
```

In [37]:
```
r.text[:2000]
```

Out[37]:
```
u'{"info": {"num_results": 417, "limit": 100, "offset": 0, "query": "kaizers orchestra", "type": "track", "page": 1},
"tracks": [{"album": {"released": "2012", "href": "spotify:album:5AN6A9IR1g1xRgY0RoKOsT", "name": "Hjerteknuser",
"availability": {"territories": "NO"}}, "name": "Hjerteknuser", "popularity": "0.66621", "external-ids": [{"type":
"isrc", "id": "NOHDL1002070"}], "length": 199.407, "href": "spotify:track:6dKWi7apHjn2W7Ojncv4Wu", "artists": [{"href":
```

```
"spotify:artist:1s1DnVoBDfp3jxjjew8cBR", "name": "Kaizers Orchestra"}], "track-number": "1"}, {"album": {"released":
"2012", "href": "spotify:album:2oZ0PnxiH9LaoUAFzlPSGK", "name": "Siste dans", "availability": {"territories": "NO"}},
"name": "Siste dans", "popularity": "0.64841", "external-ids": [{"type": "isrc", "id": "NOHDL1202060"}], "length":
217.891, "href": "spotify:track:0z26fQRDfSwxxuyKYrjZn3", "artists": [{"href": "spotify:artist:1s1DnVoBDfp3jxjjew8cBR",
"name": "Kaizers Orchestra"}], "track-number": "1"}, {"album": {"released": "2010", "href":
"spotify:album:6jbtJwRmuezfOSXyJy3tRZ", "name": "Violeta Violeta Volume I", "availability": {"territories": "NO"}},
"name": "Hjerteknuser", "popularity": "0.63483", "external-ids": [{"type": "isrc", "id": "NOHDL1002070"}], "length":
200.322, "href": "spotify:track:3NThq9BqYtKYBfsHIogjM6", "artists": [{"href": "spotify:artist:1s1DnVoBDfp3jxjjew8cBR",
"name": "Kaizers Orchestra"}], "track-number": "7"}, {"album": {"released": "2012", "href":
"spotify:album:5E9Kg0KC7H0CWOVongiKRe", "name": "Violeta Violeta Volume III", "availability": {"territories": "NO"}},
"name": "Begravelsespolka", "popularity": "0.61493", "external-ids": [{"type": "isrc", "id": "NOHDL1202020"}], "length":
426.58, "href": "spotify:track:4k2VAoUhJx7lxMscgY8USe", "artists": [{"href": "spotify:artist:1s1DnVoBDfp3jxjjew8cBR",
"name": "Kaizers Orchestra"}], "track-number": "1"}, {"album": {"released": "2012", "href":
"spotify:album:0hoeWFBKo9kGoKMhKOuKRY", "name": "V\\u00e5re Demoner", "availabilit'
```

In [38]:
```python
json_data = r.json()
```

In [39]:
```python
len(json_data["tracks"])
```

Out[39]:
```
100
```

In [40]:
```python
json_data["tracks"][1]
```

Out[40]:
```
{u'album': {u'availability': {u'territories': u'NO'},
  u'href': u'spotify:album:2oZ0PnxiH9LaoUAFzlPSGK',
  u'name': u'Siste dans',
  u'released': u'2012'},
 u'artists': [{u'href': u'spotify:artist:1s1DnVoBDfp3jxjjew8cBR',
   u'name': u'Kaizers Orchestra'}],
 u'external-ids': [{u'id': u'NOHDL1202060', u'type': u'isrc'}],
 u'href': u'spotify:track:0z26fQRDfSwxxuyKYrjZn3',
 u'length': 217.891,
 u'name': u'Siste dans',
 u'popularity': u'0.64841',
 u'track-number': u'1'}
```

In [41]:
```python
r.headers
```

Out[41]:
```
{'access-control-allow-origin': '*',
 'age': '10058',
 'content-length': '58592',
 'content-type': 'application/json; charset=utf-8',
 'date': 'Tue, 22 Jan 2013 12:17:59 GMT',
 'expires': 'Wed, 23 Jan 2013 09:30:21 GMT',
 'last-modified': 'Mon, 21 Jan 2013 23:35:15 GMT',
 'server': 'lighttpd smisk/1.1.6',
 'vary': 'Accept-Charset',
 'via': '1.1 varnish',
 'x-varnish': '766974840 766737315'}
```

In [42]:
```python
logo = requests.get("http://www.scilifelab.se/images/logo_header.png")
```

In [43]:
```python
logo.headers
```

Out[43]:
```
{'accept-ranges': 'bytes',
 'connection': 'close',
 'content-length': '14922',
 'content-type': 'image/png',
 'date': 'Tue, 22 Jan 2013 12:13:37 GMT',
 'etag': '"172816a-3a4a-4aab46ed30140"',
 'last-modified': 'Wed, 17 Aug 2011 14:37:17 GMT',
 'server': 'Apache/2.2.3 (CentOS)'}
```

In [44]:
```python
logo.content[:100]
```

Out[44]:
```
'\x89PNG\r\n\x1a\n\x00\x00\x00\rIHDR\x00\x00\x00\xfa\x00\x00\x00P\x08\x06\x00\x00\x000kh\xb5\x00\x00
\x00IDATx\x9c\xed\x9dw\\TW\xf6\xc0\xcf\xbd\xefMo\x94\x01\xa4\t(\xa2\x82X@\x8d1Q\x83-
\xd1\x98\xc4$\x0b\xeb\xc6\x18uctw\x13\xb3\xae\xeeoM4e\xdcMq\x93\x8dIL\xaf\x9ah'
```

In [45]:
```python
from IPython.core.display import Image
```

In [46]:
```python
Image(logo.content)
```

Out[46]:

SciLifeLab

Task

Make a module and script which fetches the XML formatted for the status of escalators in the NYC subway system at http://www.grandcentral.org/developers/data/nyct/nyct_ene.xml, calculate the fraction of those which have the reason "Repair", and prints this fraction.

(Information about the data can be found at http://www.grandcentral.org/developers/download.html)

The script should use the module, and should be installable by `python setup.py install`

```
[lastname]/
    [lastname]/
        __init__.py
    scripts/
        getting_data.py
    README.md
    setup.py
```

In `setup.py`, add the scripts to the call of the `setup` function. For details, see http://peak.telecommunity.com/DevCenter/setuptools#basic-use

This means that after `python setup.py install` has been run, it should be possible to simple type

```
$ getting_data.py
```

in the terminal and get the desired output.