Data Wrangling with MongoDB

Lesson 6: Case Study – Openstreetmap Data

Iterative parsing:
mapparser.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""
Your task is to use the iterative parsing to process the map file and
find out not only what tags are there, but also how many, to get the
feeling on how much of which data you can expect to have in the map.
Fill out the count_tags function. It should return a dictionary with the
tag name as the key and number of times this tag can be encountered in
the map as value.

Note that your code will be tested with a different data file than the 'example.osm'
"""
import xml.etree.ElementTree as ET
import pprint

def count_tags(filename):
    tags = {}
    for event,elem in ET.iterparse(filename):
        if elem.tag not in tags:
            tags[elem.tag] = 1
        else:
            tags[elem.tag] += 1

    return tags
    # YOUR CODE HERE
def test():

    tags = count_tags('example.osm')
    pprint.pprint(tags)
    assert tags == {'bounds': 1,
                    'member': 3,
                    'nd': 4,
                    'node': 20,
                    'osm': 1,
                    'relation': 1,
                    'tag': 7,
                    'way': 1}
```

```
if __name__ == "__main__":
    test()
```

mapparser.py Output:

```
{'bounds': 1,
 'member': 3,
 'nd': 4,
 'node': 20,
 'osm': 1,
 'relation': 1,
 'tag': 7,
 'way': 1}
Your program counted the correct number of 'node' elements!
Your program counted the correct number of 'way' elements!
Your program counted the correct number of 'tag' elements!
```

Data Model:

We would like to get the data into a database.
But we need to decide on a data model!
Which of the following models would you prefer
for the following data node?


```
{
"_id": "2406124091",
"visible":"true",
"version":"2",
"changeset":"17206049",
"timestamp":"2013-08-03T16:43:42Z",
"user":"linuxUser16",
"uid":"1219059",
"pos": [41.9757030, -87.6921867],
"tags":[ {"k": "addr:housenumber", "v": "5157"},
         {"k": "addr:postcode", "v": "60625"},
         {"k": "addr:street", "v": "North Lincoln Ave"},
         {"k": "amenity", "v": "restaurant"},
       ]
}
```

```
{
"_id": "2406124091",
"visible":"true",
"created": {
          "version":"2",
          "changeset":"17206049",
          "timestamp":"2013-08-03T16:43:42Z",
          "user":"linuxUser16",
          "uid":"1219059"
        },
"pos": [41.9757030, -87.6921867],
"address": {
          "housenumber": "5157",
          "postcode": "60625",
          "street": "North Lincoln Ave"
        },
"amenity": "restaurant",
"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}
```

Tag Types:

tags.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.ElementTree as ET
import pprint
import re
"""
Your task is to explore the data a bit more.
Before you process the data and add it into MongoDB, you should
check the "k" value for each "<tag>" and see if they can be valid keys in MongoDB,
as well as see if there are any other potential problems.

We have provided you with 3 regular expressions to check for certain patterns
in the tags. As we saw in the quiz earlier, we would like to change the data model
and expand the "addr:street" type of keys to a dictionary like this:
{"address": {"street": "Some value"}}
So, we have to see if we have such tags, and if we have any tags with problematic characters.
Please complete the function 'key_type'.
"""
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

def key_type(element, keys):
    if element.tag == "tag":
        print element.attrib['k']
        if re.match(problemchars,element.attrib['k']) or re.search(r'\?',element.attrib['k']):
            keys['problemchars'] += 1
        elif re.match(lower,element.attrib['k']):
            keys['lower'] += 1
        elif re.match(lower_colon,element.attrib['k']):
            keys['lower_colon'] += 1
        else:
            keys['other'] += 1

    return keys

def process_map(filename):
    keys = {"lower": 0, "lower_colon": 0, "problemchars": 0, "other": 0}
    for _, element in ET.iterparse(filename):
        keys = key_type(element, keys)

    return keys

def test():
    # You can use another testfile 'map.osm' to look at your solution
    # Note that the assertions will be incorrect then.
```

```
    keys = process_map('example.osm')
    pprint.pprint(keys)
    assert keys == {'lower': 5, 'lower_colon': 0, 'other': 1, 'problemchars': 1}

if __name__ == "__main__":
    test()
```

tags.py Output:

```
highway
amenity?
cuisine
NAME
highway
restriction
type
{'lower': 5, 'lower_colon': 0, 'other': 1, 'problemchars': 1} You successfully found the correct tag type counts!
```

users.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.ElementTree as ET
import pprint
import re
"""
Your task is to explore the data a bit more.
The first task is a fun one - find out how many unique users
have contributed to the map in this particular area!

The function process_map should return a set of unique user IDs ("uid")
"""
def get_user(element):
    return

def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if 'uid' in element.attrib:
            users.add(element.attrib['uid'])

    return users


def test():
```

```
    users = process_map('example.osm')
    pprint.pprint(users)
    assert len(users) == 6

if __name__ == "__main__":
    test()
```

users.py Output:

```
set(['1219059', '147510', '26299', '451048', '567034', '939355'])
You successfully found the correct number of unique users!
```

Improving Street Names:

audit.py

```
"""
Your task in this exercise has two steps:

- audit the OSMFILE and change the variable 'mapping' to reflect the changes needed to fix
    the unexpected street types to the appropriate ones in the expected list.
    You have to add mappings only for the actual problems you find in this OSMFILE,
    not a generalized solution, since that may and will depend on the particular area you are auditing.
- write the update_name function, to actually fix the street name.
    The function takes a string with street name as an argument and should return the fixed name
    We have provided a simple test so that you see what exactly is expected
"""
import xml.etree.cElementTree as ET
from collections import defaultdict
import re
import pprint

OSMFILE = "example.osm"
street_type_re = re.compile(r'\b\S+\.?$', re.IGNORECASE)

expected = ["Street", "Avenue", "Boulevard", "Drive", "Court", "Place", "Square", "Lane", "Road",
            "Trail", "Parkway", "Commons"]

# UPDATE THIS VARIABLE
mapping = { "St": "Street",
            "St.": "Street",
            "Ave" : "Avenue",
            "Rd." : "Road",
            "N."  : "North"
          }
```

```
def audit_street_type(street_types, street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type not in expected:
            street_types[street_type].add(street_name)


def is_street_name(elem):
    return (elem.attrib['k'] == "addr:street")


def audit(osmfile):
    osm_file = open(osmfile, "r")
    street_types = defaultdict(set)
    for event, elem in ET.iterparse(osm_file, events=("start",)):

        if elem.tag == "node" or elem.tag == "way":
            for tag in elem.iter("tag"):
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])

    return street_types

def update_name(name, mapping):

    if name.split()[-1] in mapping:
        name = name.replace(name.split()[-1], mapping[name.split()[-1]])
    if name.split()[0] in mapping:
        name = name.replace(name.split()[0], mapping[name.split()[0]])

    return name

def test():
    st_types = audit(OSMFILE)
    assert len(st_types) == 3
    pprint.pprint(dict(st_types))

    for st_type, ways in st_types.iteritems():
        for name in ways:
            better_name = update_name(name, mapping)
            print name, "=>", better_name
            if name == "West Lexington St.":
                assert better_name == "West Lexington Street"
            if name == "Baldwin Rd.":
                assert better_name == "Baldwin Road"

if __name__ == '__main__':
```

```
test()
```

audit.py Output:

```
{'Ave': set(['N. Lincoln Ave', 'North Lincoln Ave']),
 'Rd.': set(['Baldwin Rd.']),
 'St.': set(['West Lexington St.'])}
N. Lincoln Ave => North Lincoln Avenue
North Lincoln Ave => North Lincoln Avenue
West Lexington St. => West Lexington Street
Baldwin Rd. => Baldwin Road
All test cases were processed correctly!
```

Preparing for Database:

data.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
import xml.etree.ElementTree as ET
import pprint
import re
import codecs
import json
"""
Your task is to wrangle the data and transform the shape of the data
into the model we mentioned earlier. The output should be a list of dictionaries
that look like this:

{
"id": "2406124091",
"type: "node",
"visible":"true",
"created": {
        "version":"2",
        "changeset":"17206049",
        "timestamp":"2013-08-03T16:43:42Z",
        "user":"linuxUser16",
        "uid":"1219059"
      },
"pos": [41.9757030, -87.6921867],
"address": {
        "housenumber": "5157",
        "postcode": "60625",
        "street": "North Lincoln Ave"
      },
"amenity": "restaurant",
```

"cuisine": "mexican",
"name": "La Cabana De Don Luis",
"phone": "1 (773)-271-5176"
}

You have to complete the function 'shape_element'.
We have provided a function that will parse the map file, and call the function with the element as an argument. You should return a dictionary, containing the shaped data for that element.
We have also provided a way to save the data in a file, so that you could use mongoimport later on to import the shaped data into MongoDB.

Note that in this exercise we do not use the 'update street name' procedures you worked on in the previous exercise. If you are using this code in your final project, you are strongly encouraged to use the code from previous exercise to update the street names before you save them to JSON.

In particular the following things should be done:
- you should process only 2 types of top level tags: "node" and "way"
- all attributes of "node" and "way" should be turned into regular key/value pairs, except:
    - attributes in the CREATED array should be added under a key "created"
    - attributes for latitude and longitude should be added to a "pos" array,
      for use in geospacial indexing. Make sure the values inside "pos" array are floats
      and not strings.
- if second level tag "k" value contains problematic characters, it should be ignored
- if second level tag "k" value starts with "addr:", it should be added to a dictionary "address"
- if second level tag "k" value does not start with "addr:", but contains ":", you can process it
  same as any other tag.
- if there is a second ":" that separates the type/direction of a street,
  the tag should be ignored, for example:

<tag k="addr:housenumber" v="5158"/>
<tag k="addr:street" v="North Lincoln Avenue"/>
<tag k="addr:street:name" v="Lincoln"/>
<tag k="addr:street:prefix" v="North"/>
<tag k="addr:street:type" v="Avenue"/>
<tag k="amenity" v="pharmacy"/>

  should be turned into:

{...
"address": {
    "housenumber": 5158,
    "street": "North Lincoln Avenue"
}
"amenity": "pharmacy",
...
}

- for "way" specifically:

```
 <nd ref="305896090"/>
 <nd ref="1719825889"/>

should be turned into
"node_refs": ["305896090", "1719825889"]
"""
lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

CREATED = [ "version", "changeset", "timestamp", "user", "uid"]
POS = [ "lat", "lon" ]

def shape_element(element):
   node = {}
   node['created'] = {}
   node['pos'] = [12345,12345]
   node['address'] = {}
   node['node_refs'] = []

   if element.tag == "node" or element.tag == "way" :

      if element.tag == 'node':
         node['type'] = 'node'
         for attrib in element.attrib:
            if attrib not in CREATED and attrib not in POS:
               node[attrib] = element.attrib[attrib]
            elif attrib in CREATED:
               node['created'][attrib] = element.attrib[attrib]
            elif attrib in POS:
               if attrib == 'lat':
                  node['pos'][0] = float(element.attrib[attrib])
               elif attrib == 'lon':
                  node['pos'][1] = float(element.attrib[attrib])

         for child in element:
            if 'addr:' in child.get('k') and (child.get('k').count(':') == 1) and not
re.match(problemchars,child.get('k')):
               node['address'][child.get('k').replace('addr:','')] = child.get('v')
            elif 'addr:' not in child.get('k'):
               node[child.get('k')] = child.get('v')

      elif element.tag == 'way':
         node['type'] = 'way'
         for child in element:
            if child.tag == 'nd':
```

```
                node['node_refs'].append(child.get('ref'))
            else:
                if 'addr:' in child.get('k') and (child.get('k').count(':') == 1) and not
re.match(problemchars,child.get('k')):
                    node['address'][child.get('k').replace('addr:','')] = child.get('v')
                elif 'addr:' not in child.get('k'):
                    node[child.get('k')] = child.get('v')

        # cleanup
        if not node['created']:
            del node['created']
        if node['pos'][0] == 12345 or node['pos'][1] == 12345:
            del node['pos']
        if not node['address']:
            del node['address']
        if node['node_refs'] == []:
            del node['node_refs']

        #print node
        return node
    else:
        return None

def process_map(file_in, pretty = False):
    # You do not need to change this file
    file_out = "{0}.json".format(file_in)
    data = []
    with codecs.open(file_out, "w") as fo:
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+"\n")
                else:
                    fo.write(json.dumps(el) + "\n")
    return data

def test():
    # NOTE: if you are running this code on your computer, with a larger dataset,
    # call the process_map procedure with pretty=False. The pretty=True option adds
    # additional spaces to the output, making it significantly larger.
    data = process_map('example.osm', True)
    #pprint.pprint(data)

    correct_first_elem = {
        "id": "261114295",
        "visible": "true",
```

```
        "type": "node",
        "pos": [41.9730791, -87.6866303],
        "created": {
          "changeset": "11129782",
          "user": "bbmiller",
          "version": "7",
          "uid": "451048",
          "timestamp": "2012-03-28T18:31:23Z"
        }
    }
    assert data[0] == correct_first_elem
    assert data[-1]["address"] == {
                                "street": "West Lexington St.",
                                "housenumber": "1412"
                                  }
    assert data[-1]["node_refs"] == [ "2199822281", "2199822390",  "2199822392", "2199822369",
                                "2199822370", "2199822284", "2199822281"]

if __name__ == "__main__":
    test()
```

data.py Output:

```
All processing tests passed!
```