**Lesson 6 code**

**Iterative parsing:**

*import xml.etree.ElementTree as ET*

*import pprint*

*def count\_tags(filename):*

*dicttag = {}*

*for event, elem in ET.iterparse(filename):*

*if elem.tag in dicttag:*

*dicttag[elem.tag] += 1*

*else:*

*dicttag[elem.tag] = 1*

*return (dicttag)*

*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()*

**Tag Types**

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":

if re.match(lower, element.attrib['k']):

keys['lower'] +=1

elif re.match(lower\_colon, element.attrib['k']):

keys['lower\_colon'] += 1

elif re.match(problemchars, element.attrib['k']):

keys['problemchars'] += 1

else:

keys['other'] += 1

print keys

pass

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()

**Exploring Users**

import xml.etree.ElementTree as ET

import pprint

import re

def get\_user(element):

return

def process\_map(filename):

users = set()

for \_, element in ET.iterparse(filename, events =('start',)):

if 'uid' in element.attrib:

if element.attrib['uid'] in users:

pass

else:

users.add(element.attrib['uid'])

print users

pass

return users

def test():

users = process\_map('example.osm')

pprint.pprint(users)

assert len(users) == 6

if \_\_name\_\_ == "\_\_main\_\_":

test()

**Improving Street Names**

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"]

mapping = { "St": "Street",

"St.": "Street",

'Rd.':'Road',

'Ave':'Avenue'

}

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):

for item in mapping:

if name.find(item) > 0:

name = name[:name.find(item)] + mapping[item]

print name

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()

**Preparing for Database**

import xml.etree.ElementTree as ET

import pprint

import re

import codecs

import json

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"]

def shape\_element(element):

node = {}

createdtag = {}

address = {}

pos = [0,0]

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

for tag in CREATED:

createdtag[tag] = element.attrib[tag]

node['created'] = createdtag

if 'lat' and 'lon' in element.attrib:

pos[0] = float(element.attrib['lat'])

pos[1] = float(element.attrib['lon'])

for tag in element.iter('tag'):

if tag.attrib['k'].count(':') >= 2:

pass

elif re.match(problemchars, tag.attrib['k']):

pass

elif tag.attrib['k'].startswith('addr:'):

address[tag.attrib['k'].split(':')[1]] = tag.attrib['v']

if tag.attrib['k'].startswith !='addr:' and tag.attrib['k'].count(':') == 1:

node[tag.attrib['k'].split(':')[1]] = tag.attrib['v']

node['address'] = address

node['pos'] = pos

node['id'] = element.attrib['id']

node['type'] = element.tag

print address

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)

assert data[0] == {

"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[-1]["address"] == {

"street": "West Lexington St.",

"housenumber": "1412"

}

assert data[-1]["node\_refs"] == [ "2199822281", "2199822390", "2199822392", "2199822369",

"2199822370", "2199822284", "2199822281"]

if \_\_name\_\_ == "\_\_main\_\_":

test()