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.

The output should be 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):

tag\_count = {}

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

add\_tag(element.tag, tag\_count)

return tag\_count

def add\_tag(tag, tag\_count):

if tag in tag\_count:

tag\_count[tag] += 1

else:

tag\_count[tag] = 1

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 Type = tagtype.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":

key = element.get("k")

print key,

# value = element.get("v")

if problemchars.search(key):

keys['problemchars'] += 1

print '--> problemchars'

elif lower\_colon.search(key):

keys['lower\_colon'] += 1

print '--> lower\_colon'

elif lower.search(key):

keys['lower'] += 1

print '--> lower'

else:

keys['other'] += 1

print '--> other'

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 = 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.get('uid'))

return users

def test():

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

pprint.pprint(users)

assert len(users) == 6

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

test()

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"

}

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):

m = street\_type\_re.search(name)

better\_name = name

if m:

better\_street\_type = mapping[m.group()]

better\_name = street\_type\_re.sub(better\_street\_type, name)

return better\_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 = 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"]

def shape\_element(element):

node = {}

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

for key in element.attrib.keys():

val = element.attrib[key]

node["type"] = element.tag

if key in CREATED:

if not "created" in node.keys():

node["created"] = {}

node["created"][key] = val

elif key == "lat" or key == "lon":

if not "pos" in node.keys():

node["pos"] = [0.0, 0.0]

old\_pos = node["pos"]

if key == "lat":

new\_pos = [float(val), old\_pos[1]]

else:

new\_pos = [old\_pos[0], float(val)]

node["pos"] = new\_pos

else:

node[key] = val

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

tag\_key = tag.attrib['k']

tag\_val = tag.attrib['v']

if problemchars.match(tag\_key):

continue

elif tag\_key.startswith("addr:"):

if not "address" in node.keys():

node["address"] = {}

addr\_key = tag.attrib['k'][len("addr:") : ]

if lower\_colon.match(addr\_key):

continue

else:

node["address"][addr\_key] = tag\_val

elif lower\_colon.match(tag\_key):

node[tag\_key] = tag\_val

else:

node[tag\_key] = tag\_val

for tag in element.iter("nd"):

if not "node\_refs" in node.keys():

node["node\_refs"] = []

node\_refs = node["node\_refs"]

node\_refs.append(tag.attrib["ref"])

node["node\_refs"] = node\_refs

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()