Project3-codes

February 3, 2016

1 These are complete lesson 6 quiz code, including codes that I added to solve the problems

In [ ]:
```
#!/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.cElementTree as ET
import pprint

def count_tags(filename):
        # YOUR CODE HERE
    tags = {}
    for event, elem in ET.iterparse(filename):
        if(elem.tag in tags):
            tags[elem.tag] += 1
        else:
            tags[elem.tag] = 1
    return tags
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()

In [ ]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        import xml.etree.cElementTree 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', such that we have a count of each of
        four tag categories in a dictionary:
          "lower", for tags that contain only lowercase letters and are valid,
          "lower_colon", for otherwise valid tags with a colon in their names,
          "problemchars", for tags with problematic characters, and
          "other", for other tags that do not fall into the other three categories.
        See the 'process_map' and 'test' functions for examples of the expected format.
        """


        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":
                # YOUR CODE HERE
                if (lower.search(element.attrib['k'])):
                    keys["lower"] += 1
                elif(lower_colon.search(element.attrib['k'])):
                    keys['lower_colon'] += 1
                elif(problemchars.search(element.attrib['k'])):
                    keys['problemchars'] += 1
                else:
                    keys['other'] += 1
                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 assertion below will be incorrect then.
        # Note as well that the test function here is only used in the Test Run;
        # when you submit, your code will be checked against a different dataset.
        keys = process_map('example.osm')
        pprint.pprint(keys)
        assert keys == {'lower': 5, 'lower_colon': 0, 'other': 1, 'problemchars': 1}


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

```
In [ ]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        import xml.etree.cElementTree 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(element.tag == 'node' or element.tag == 'way' or element.tag == 'relation'):
                    users.add(element.attrib['uid'])
                pass
            return users


        def test():

            users = process_map('example.osm')
            pprint.pprint(users)
            assert len(users) == 6
        if __name__ == "__main__":
            test()

In [ ]: """
        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 a
        - 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 nam
            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",
                    "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):

    # YOUR CODE HERE
    name = name.rsplit(' ', 1)[0] + " " + mapping[name.rsplit(' ', 1)[1]]
    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()
```

```
In [ ]: #!/usr/bin/env python
        # -*- coding: utf-8 -*-
        import xml.etree.cElementTree 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 elemen
        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 "addres
        - if second level tag "k" value does not start with "addr:", but contains ":", you can process
          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" :
        # YOUR CODE HERE
        node['type'] = element.tag
        node['created'] = {}
        node['pos'] = []
        for attribute, val in element.attrib.iteritems():
            if(attribute in CREATED):
                node['created'][attribute] = val
            elif(attribute in ['lat', 'lon']):
                node['pos'].append(float(val))
            else:
                node[attribute] = val
        for tag in element.iter("tag"):
            attrib_val = tag.attrib['k']
            if( not (problemchars.search(attrib_val)) and attrib_val.count(":") < 2):
                if(attrib_val.startswith("addr:")):
                    if('address' not in node):
                        node['address'] = {}
                    node['address'][attrib_val[5:]] = tag.attrib['v']
                elif(attrib_val.find(":") != -1):
                    node[attrib_val] = tag.attrib['v']
        print node
        return node
    else:
        return None


def process_map(file_in, pretty = False):
    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():
        data = process_map('example.osm', True)

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

2 These are codes used for auditing and processing data for final project

2.1 Code for extracting a sample of 1/10 of the size of actual data set

```
In [ ]: import xml.etree.cElementTree as ET
        import pprint

        SAMPLE_FILE = "toronto_sample.osm"

        def get_element(osm_file, tags=('node', 'way', 'relation')):
            """Yield element if it is the right type of tag

            Reference:
            http://stackoverflow.com/questions/3095434/inserting-newlines-in-xml-file-generated-via-xml
            """
            context = ET.iterparse(osm_file, events=('start', 'end'))
            _, root = next(context)
            for event, elem in context:
                if event == 'end' and elem.tag in tags:
                    yield elem
                    root.clear()

        with open(SAMPLE_FILE, 'wb') as output:
            output.write('<?xml version="1.0" encoding="UTF-8"?>\n')
            output.write('<osm>\n  ')

            # Write every 10th top level element
            for i, element in enumerate(get_element("toronto_sample.osm")):
                if i % 10 == 0:
                    output.write(ET.tostring(element, encoding='utf-8'))

            output.write('</osm>')
```

2.2 This is the code used to put the cleaned data into a JSON file:

```
In [ ]: import xml.etree.cElementTree as ET
        import pprint
        import re
        import codecs
```

```python
import json

def update_name(name, mapping):
    name = name.rsplit(' ', 1)[0] + " " + mapping[name.rsplit(' ', 1)[1]]
    return name

#cleaning the data and putting it in json
def change_street_type(street_name):
    m = street_type_re.search(street_name)
    if m:
        street_type = m.group()
        if street_type in mapping:
            return update_name(street_name, mapping)
        else:
            return street_name

#example: change "city of Burlington" to "Burlington"
def edit_city_name(city_name):
    if (city_name.startswith("City of") or city_name.startswith("city of")\
        or city_name.startswith("Town of") or city_name.startswith("town of")):
        return city_name.split()[2]
    else:
        return city_name

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" :
        node['type'] = element.tag
        node['created'] = {}
        node['pos'] = []
        #iterate over all attributes of the element and put them in a JSON dictionary
        for attribute, val in element.attrib.iteritems():
            if(attribute in CREATED):
                node['created'][attribute] = val
            elif(attribute in ['lat', 'lon']):
                node['pos'].append(float(val))
            else:
                node[attribute] = val
        for tag in element.iter("tag"):
            attrib_val = tag.attrib['k']
            if( not (problemchars.search(attrib_val)) and attrib_val.count(":") < 2):
                if(attrib_val.startswith("addr:")):
                    if('address' not in node):
                        node['address'] = {}
                    if attrib_val == "addr:street":
                        node['address'][attrib_val[5:]] = change_street_type(tag.attrib['v'])
                    if attrib_val == "addr:city":
                        node['address'][attrib_val[5:]] = edit_city_name(tag.attrib['v'])
                elif(attrib_val.find(":") != -1):
                    node[attrib_val] = tag.attrib['v']
                else:
                    if('other_tags' not in node):
                        node['other_tags'] = {}
                    #put all other tags in a dictionary under the name of "other_tags"
                    node['other_tags'][attrib_val] = tag.attrib['v']
        return node
    else:
        return None


def process_map(file_in, pretty = False):
    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():
    data = process_map('toronto_sample.osm', False)

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

```
In [ ]: #This code is written to compare contents of all same tags to see if there is any inconsistanci
        import xml.etree.cElementTree as ET
        import pprint

        def count_tags(filename):
                # YOUR CODE HERE
            tags = {}
            count = 0
            for event, elem in ET.iterparse(filename, events=("start",)):
                if elem.tag not in tags:
                    tags[elem.tag] = {}
                    for tag in elem.iter():
                        if tag.tag not in tags[elem.tag]:
                            tags[elem.tag][tag.tag]  = {}
                        for attribute, val in tag.attrib.iteritems():
                            if attribute not in tags[elem.tag][tag.tag]:
                                tags[elem.tag][tag.tag][attribute] = []
                                tags[elem.tag][tag.tag][attribute].append(val)
                            else:
                                tags[elem.tag][tag.tag][attribute].append(val)
            return tags
        def test():
            tags = count_tags('toronto_sample.osm')
            pprint.pprint(tags)

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

```
In [ ]: #get number of unique users using Python code
        def process_map(filename):
            users = set()
            for _, element in ET.iterparse(filename):
                if(element.tag == 'node' or element.tag == 'way' or element.tag == 'relation'):
                    users.add(element.attrib['uid'])
                pass
            return users


        def test():

            users = process_map('toronto_sample.osm')
            pprint.pprint(users)
            print "Number of unique users:: ", len(users)

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

2.3 I also looked into relation elements

```
In [ ]: def shape_element(element):
            node = {}
            if element.tag == "relation" :
                node['type'] = element.tag
                for tag in element.iter():
                    print tag.tag
                    for key, val in tag.attrib.iteritems():
                        print "key: ", key , " val: ", val
                return node
            else:
                return None


        def process_map(file_in, pretty = False):
            for _, element in ET.iterparse(file_in):
                el = shape_element(element)

        def test():
            data = process_map('toronto_sample.osm', False)

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

2.4 Then I started with MongoDb

```
In [ ]: #add site_packages, I already had Ipython Notebook installed in another directory.
        sys.path.append("C:\\Users\\owner\\Anaconda3\\Lib\\site-packages")
        import pymongo

        def get_db(db_name):
            from pymongo import MongoClient
            client = MongoClient('localhost:27017')
            db = client[db_name]
            return db

        db = get_db('cities')
```

```
In [ ]: #Top 10 sources
        sources = db.toronto.aggregate([{"$match":{"other_tags.source":{"$exists":1}}},
                                        {"$group":{"_id":"$other_tags.source", "count":{"$sum":1}}},
                                                          {"$sort":{"count":-1}}, {"$limit":10}])
        for source in sources:
            print source
```

```
In [ ]: #Top 10 users
        users = db.toronto.aggregate([{"$project":{"user":"$created.user","source":"$other_tags.source"
                                      {"$group":{"_id":"$user", "count":{"$sum":1}}},
                                      {"$project":{"count":"$count","user":"$created.user","source":"$
                                       {"$sort":{"count":-1}}, {"$limit":10}])
        for user in users:
            print user
```

```
In [ ]: #top 10 religious amenities
        religions = db.toronto.aggregate([{"$match":{"other_tags.amenity":{"$exists":1}, "other_tags.am
                                          {"$group":{"_id":"$other_tags.religion", "count":{"$sum":1}
                                          {"$sort":{"count":-1}}, {"$limit":10}])

        for religion in religions:
            print religion
```

```
In [ ]: #top 10 leisure centers
        leisures = db.toronto.aggregate([{"$match":{"other_tags.leisure":{"$exists":1}}},

        {"$group":{"_id":"$other_tags.leisure", "count":{"$sum":1}}},

        {"$sort":{"count":-1}}, {"$limit":10}])

        for leisure in leisures:
            print leisure
```

2.5 Calculating proportion of roads with bicycle lane (the next 3 cells)

```
In [ ]: all_roads = db.toronto.aggregate([{"$match":{"other_tags.surface":"asphalt"}},
                                         {"$group":{"_id":"$other_tags.surface", "count":{"$sum":1}}}])
```

```
In [ ]: bicycle = db.toronto.aggregate([{"$match":{"other_tags.bicycle":"yes"}},
                                         {"$group":{"_id":"$other_tags.bicycle", "count":{"$sum":1}}}])
```

```
In [ ]: print float(list(bicycle)[0]['count'])/float(list(all_roads)[0]['count'])
```

```
In [ ]: #Top 10 sport aminities. I thought there would be many more hokey rinks.
        sports = db.toronto.aggregate([{"$match":{"other_tags.sport":{"$exists":"true"}}},
                                         {"$group":{"_id":"$other_tags.sport", "count":{"$sum":1}}},{"$s
        for sport in sports:
            print sport
```

```
In [ ]: #Number of node elements in sample dataset
        db.toronto.find({"type":"node"}).count()
```

```
In [ ]: #Number of way elements in sample dataset
        db.toronto.find({"type":"way"}).count()
```

```
In [ ]: #Number of unique users
        len(db.toronto.distinct("created.user"))
```

```
In [6]: import os
        os.environ['PATH']
```

```
Out[6]: 'C:\\ProgramData\\Oracle\\Java\\javapath;C:\\Program Files (x86)\\Intel\\iCLS Client\\;C:\\Progr
```

```
In [ ]:
```