Project Code Snippets

P1

Auditing the data

```
import xml.etree.ElementTree as ET
import pprint
tags = {}
for event, elem in ET.iterparse(filename):
if elem.tag in tags:
tags[elem.tag] += 1
else:
tags[elem.tag] = 1
pprint.pprint(tags)
```

With the result being:

```
{'bounds': 1,
 'member': 13056,
 'nd': 889130,
 'node': 768631,
 'osm': 1,
 'relation': 1283,
 'tag': 535875,
 'way': 79692}
```

P2

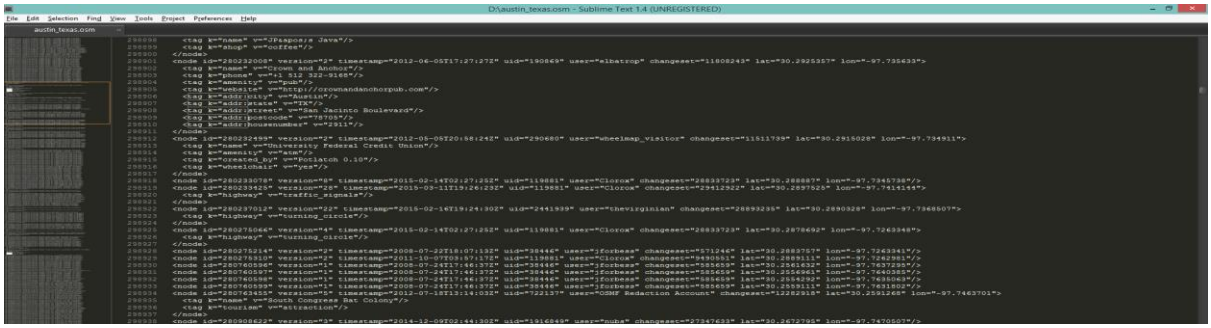
This is where there are variables declared to search for the following three regular expressions: **lower**, **lower_colon**, and **problemchars**

lower = re.compile(r'^([a-z]|_)*$')

lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')

problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')

P3



P4

```
def process_map(filename):
    users = set()
    for _, element in ET.iterparse(filename):
        if "uid" in element.attrib and element.tag in ('node', 'way'):
```

```
            users.add(element.attrib["uid"])
    return users


def test():
    users = process_map('D:\\austin_texas.osm')

    print len(users)
```

The output returned was: 910

P5

Building a regex to match the last token in a string optionally ending with a period and a list of street types to clean found in P4.:

from collections import defaultdict

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

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

P6

Used to hold expected dictionaries it will be added to it ..:

```
def audit_string(match_set_dict, string_to_audit, regex, expected_matches):
    m = regex.search(string_to_audit)
    if m:
        match_string = m.group()
        if match_string not in expected_matches:
            match_set_dict[match_string].add(string_to_audit)
```

P7

```
def audit(osmfile, tag_filter, regex, expected_matches = []):
    osm_file = open(osmfile, "r")
    match_sets = defaultdict(set)
    # iteratively parse the mapping xml
    for event, elem in ET.iterparse(osm_file, events=("start",)):
        # node and way tags are of special interest
        if elem.tag == "node" or elem.tag == "way":
            # iterate the "tag" tags within a node or way
            for tag in elem.iter("tag"):
                if tag_filter(tag):
                    audit_string(match_sets, tag.attrib['v'], regex,
expected_matches)
    return match_sets
```

The function **is_street_name** determines if an element contains an attribute **k="addr:street".** I will use **is_street_name** as the **tag_filter** when I call the audit function to audit street names.

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

P8

```
street_types = audit(OSMFILE, tag_filter = is_street_name, regex = street_type_re,
expected_matches = expected)
pprint.pprint(dict(street_types))
```

Results Summary:

```
{'100': set(['Jollyville Road Suite 100', 'Old Jollyville Road, Suite 100']),
 '101': set(['4207 James Casey st #101']),
 '104': set(['11410 Century Oaks Terrace Suite #104']),
 '1100': set(['Hwy 290 W, Bldg 1100']),
 '12': set(['Ranch to Market Road 12']),
 '120': set(['Building B Suite 120']),…
```

P9

```
def update(string_to_update, mapping, regex):
    m = regex.search(string_to_update)
    if m:
        match = m.group()
        if match in mapping:
            string_to_update = re.sub(regex, mapping[match], string_to_update)
    return string_to_update
```

The results of the audit function were used to build a dictionary to map abbreviations to their full and clean representation.

```
map_street_types = \
{
"Ave" : "Avenue",
"BLVD" : "Boulevard",
"Blvd" : "Boulevard",
"Blvd." : "Boulevard",
"Cir" : "Circle",
"Dr" : "Drive",
"Ln" : "Lane",
"Ln" : "Lane",
"Pkwy" : "Parkway",
"Rd" : "Road",
"Rd." : "Road",
"St" : "Street",
"St." : "Street"
}
```

P10

```
from datetime import datetime
def modify_element(element):
    node = {}
    TIME_AND_USER_TAGS = ["version", "user", "uid","timestamp","changeset"]
    UNDERSCORE_RELATED_TAGS = ["barrier", "highway",
"amenity","emergency","source","shop"]
```

```
    if element.tag == "node" or element.tag == "way" :
        node['type'] = element.tag

        # Parse attributes
        for a in element.attrib:
            if a in TIME_AND_USER_TAGS:
                if 'created' not in node:
                    node['created'] = {}

                if a == "timestamp":
                    node['created'][a] = datetime.strptime(element.attrib[a], '%Y-
%m-%dT%H:%M:%SZ')
                else:
                    node['created'][a] = element.attrib[a]

            # Parse coordinates
            elif a in ['lat', 'lon']:
                if 'pos' not in node:
                    node['pos'] = [None, None]

                if a == 'lat':
                    node['pos'][0] = float(element.attrib[a])
                else:
                    node['pos'][1] = float(element.attrib[a])

            else:
                node[a] = element.attrib[a]

        # Iterate tag children
        for tag in element.iter("tag"):
            #Get the k tags from the dictionary
            if tag.attrib['k'] in UNDERSCORE_RELATED_TAGS:
                #Set the value of the noted k tag with underscores present in the
data
                tag.attrib['v'] = tag.attrib['v'].replace("_"," ")

            if not problemchars.search(tag.attrib['k']):
                # Tags with single colon and beginning with addr
                if lower_colon.search(tag.attrib['k']) and
tag.attrib['k'].find('addr') == 0:
                    if 'address' not in node:

                        node['address'] = {}

                    sub_attr = tag.attrib['k'].split(':', 1)

                    if is_street_name(tag):

                        # Do some cleaning
                        better_name = update(tag.attrib['v'], map_street_types,
street_type_updater_re)
                        best_name = update(better_name, map_cardinal_directions,
cardinal_dir_updater_re)

                        node['address'][sub_attr[1]] = best_name
                    else:
                        node['address'][sub_attr[1]] = tag.attrib['v']
```

```
                # All other tags that don't begin with "addr"
                elif not tag.attrib['k'].find('addr') == 0:
                    if tag.attrib['k'] not in node:
                        node[tag.attrib['k']] = tag.attrib['v']
                else:
                    node["tag:" + tag.attrib['k']] = tag.attrib['v']


        # Iterate nd children building a list
        for nd in element.iter("nd"):
            if 'node_refs' not in node:
                node['node_refs'] = []

            node['node_refs'].append(nd.attrib['ref'])

        return node
    else:
        return None
```

P11

```
def process_map(file_in, pretty = False):

    file_out = "{0}.json".format(file_in)

    with open(file_out, "wb") as fo:
        for _, element in ET.iterparse(file_in):
            el = modify_element(element)
            if el:
                if pretty:
                    fo.write(json.dumps(el, indent=2,
default=json_util.default)+"\n")
                else:
                    fo.write(json.dumps(el, default=json_util.default) + "\n")
```