```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from pprint import pprint
import codecs
import json
import phonenumbers
import re
import xml.etree.ElementTree as ET

import audit
'''
Builds JSON file from OSM. Parses, cleans, and shapes data accordingly.
'''

DEBUG = True

if DEBUG:
   OSMFILE = 'data/austin-subset.osm'
else:
   OSMFILE = 'data/austin.osm'

lower = re.compile(r'^([a-z]|_)*$')
lower_colon = re.compile(r'^([a-z]|_)*:([a-z]|_)*$')
problemchars = re.compile(r'[=\+/&<>;\'"\?%#$@\,\. \t\r\n]')
address_regex = re.compile(r'^addr\:')
street_regex = re.compile(r'^street')
gnis_regex = re.compile(r'^gnis\:')

CREATED_ATTRIBUTES = ['version', 'changeset', 'timestamp', 'user', 'uid']
POSITION_ATTRIBUTES = ['lat', 'lon']
IGNORED_TAGS = ['gnis:ST_num', 'text', 'tiger:Name', 'gnis:id', 'is_in',
   'gnis:feature_type', 'lake:surface_area:acres', 'gnis:county_id', 'iata',
   'stop', 'trees', 'icao', 'gnis:County', 'gnis:county_num', 'name:en', 'gnis:state_id',
   'health_specialty:palliative_medicine', 'tiger:STATEFP', 'name:ru', 'name:uk'
   'wikipedia:en', 'Hardware Store', 'isced', 'reg_ref', 'start_date',
   'reg_name', 'al', 'isced:level', 'source:maxspeed', 'gnis_state_id',
   'undefined', 'int_ref', 'source:ref:note', 'gnis:ST_alpha', 'gnis:feature_id',
   'practice', 'lake:shore_length:miles', 'gnis:edited', 'gnis:freature_id',
   'name:ar', 'cycleway:left', 'import_uuid', 'odbl:note', 'is_in:state',
   'gnis:reviewed', 'name:backward', 'gnis:fcode', 'is_in:country_code',
   'is_in:iso_3166_2', 'name:brand', 'name:pl', 'gnis:st_alpha']
ALIAS_TAGS = ['name_1', 'old_name', 'alt_name', 'name_2', 'place_name', 'loc_name',
   'official_name', 'name_3', 'short_name', 'bridge_name']
ZIPCODE_TAGS = ['addr:postcode', 'tiger:zip_left', 'tiger:zip_left_1', 'tiger:zip_left_2',
   'tiger:zip_left_3', 'tiger:zip_left_4', 'tiger:zip_right', 'tiger:zip_right_1',
   'tiger:zip_right_2', 'tiger:zip_right_3', 'tiger:zip_right_4']
MAPPED_TAGS = {'cosntruction': 'construction', 'construciton': 'construction',
   'EXit_to': 'exit_to', 'note:ref': 'comment', 'source:note': 'source',
   'exit_to:left': 'exit_to', 'exit_to:right': 'exit_to', 'phone': 'contact:phone',
   'maxspeed:forward': 'maxspeed'}

def shape_element(element):
   node = {}
   created_attributes = CREATED_ATTRIBUTES
   position_attributes = POSITION_ATTRIBUTES
   ignored_tags = IGNORED_TAGS
   alias_tags = ALIAS_TAGS
   zipcode_tags = ZIPCODE_TAGS
   mapped_tags = MAPPED_TAGS


   if element.tag == 'node' or element.tag == 'way':
       # populate tag type
       node['type'] = element.tag

       # initialize specialized combination fields
       address = {}
       zipcodes = set()

       # parse through attributes
       for attribute in element.attrib:
           if attribute in created_attributes:
               if 'created' not in node:
                   node['created'] = {}
               node['created'][attribute] = element.get(attribute)
           elif attribute in position_attributes:
               continue
           else:
               node[attribute] = element.get(attribute)

       # populate position
       if 'lat' in element.attrib and 'lon' in element.attrib:
           node['pos'] = [float(element.get('lat')), float(element.get('lon'))]

       # parse second-level tags
       for child in element:
           # parse second-level tags for ways and populate `node_refs`
           if child.tag == 'nd':
               if 'node_refs' not in node:
                   node['node_refs'] = []
               if 'ref' in child.attrib:
                   node['node_refs'].append(child.get('ref'))

           # throw out not-tag elements and elements without `k` or `v`
           if child.tag != 'tag'\
           or 'k' not in child.attrib\
           or 'v' not in child.attrib:
               continue
           key = child.get('k').lower()
           val = child.get('v')

           # skip problematic characters
           if problemchars.search(key):
               continue

           # skip any gnis tags
           if gnis_regex.search(key):
               continue

           # skip ignored tags
           if key in ignored_tags:
               continue

           # swap keys for corrections
           if key in mapped_tags:
               key = mapped_tags[key]

           # extract any zip codes
           if key in zipcode_tags:
               for zipcode in process_zipcode(val):
                   zipcodes.add(zipcode)

           # set all states to TX
           if key == 'addr:state':
               key = 'TX'

           # fix and standardize phone numbers using phonenumbers module and list comprehensions
           if key == 'contact:phone':
               phone_number_matches = [m.number for m in phonenumbers.PhoneNumberMatcher(val,
"US")]
               val = ';'.join([phonenumbers.format_number(phone_number_match,
                   phonenumbers.PhoneNumberFormat.NATIONAL)
                   for phone_number_match in phone_number_matches])

           # parse address k-v pairs
           if address_regex.search(key):
               key = key.replace('addr:', '')
               address[key] = val
               continue

           # parse alias tags
           if key in alias_tags:
               if 'aliases' not in node:
                   node['aliases'] = {}
```

```
        node['aliases'][key] = val
        continue

    # parse branched tags
    if ':' in key:
        add_branched_item(key, val, node)
        continue

    # catch-all
    if key not in node:
        node[key] = val

# name fallback to aliases in priority order
if 'name' not in node and 'aliases' in node:
    for alias in alias_tags:
        if alias in node['aliases']:
            node['name'] = alias
            break

# add zipcodes field
if zipcodes:
    node['zipcodes'] = list(zipcodes)

# compile address
if len(address) > 0:
    node['address'] = {}
    street_full = None
    street_dict = {}
    street_format = ['prefix', 'name', 'type']
    # parse through address objects
    for key in address:
        val = address[key]
        if street_regex.search(key):
            if key == 'street':
                street_full = audit.clean_street_address(val)
            elif 'street:' in key:
                street_dict[key.replace('street:', '')] = val
        else:
            node['address'][key] = val

    # assign street_full or fallback to compile street dict
    if street_full:
        node['address']['street'] = street_full
    elif len(street_dict) > 0:
        unclean_street = ' '.join([street_dict[key] for key in street_format])
        node['address']['street'] = audit.clean_street_address(unclean_street)

return node
```

```
    else:
        return None

def add_branched_item(key, val, node):
    """ """
    key_split = key.split(':')
    base = key_split.pop(0)
    remainder = ':'.join(key_split)
    if type(node) == dict:
        if len(key_split) == 0:
            node[base] = val
        else:
            if base not in node:
                node[base] = {}
            add_branched_item(remainder, val, node[base])

def process_zipcode(string):
    result = []
    groups = [group.strip() for group in string.split(';')]
    for group in groups:
        if re.match(r'\d{5}\:\d{5}', group):
            group_range = map(int, group.split(':'))
            result += list(map(str, range(group_range[0], group_range[1]+1)))
        elif re.match(r'\d{5}', group):
            result.append(group)
    return result

def process_map(file_in, pretty=False):
    file_out = '{0}.json'.format(file_in)
    data = []
    debug_counter = 0
    with codecs.open(file_out, 'w') as fo:
        fo.write('[\n')
        for _, element in ET.iterparse(file_in):
            el = shape_element(element)
            # if el and len(el) > 4:
            if el:
                data.append(el)
                if pretty:
                    fo.write(json.dumps(el, indent=2)+',\n')
                else:
                    fo.write(json.dumps(el) + ',\n')
                debug_counter += 1
            if debug_counter >= 10 and DEBUG:
                break
        fo.write('{}]\n')
    return data
```

```
def main():
    data = process_map(OSMFILE, pretty=DEBUG)
    # pprint(data)

def test_branched():
    node = {'tiger': {'zip_left': '43210'}}
    key = 'tiger:zip_right'
    val = '01234'
    add_branched_item(key, val, node)
    pprint(node)
    assert node == {'tiger': {'zip_left': '43210', 'zip_right': '01234'}}

def test_zipcode():
    string = "78727; 78727:78729"
    zipcodes = process_zipcode(string)
    print zipcodes
    assert zipcodes == ['78727', '78727', '78728', '78729']

if __name__ == '__main__':
    # test_branched()
    # test_zipcode()
    main()
```