```
"""
Exports:
   - `data/austin.osm-street-audit.json`
   - `data/austin.osm-zipcode-audit.json`
for use in exploratory data analysis in preparation for data cleaning.
"""
from collections import defaultdict
from pprint import pprint
import json
import re
import xml.etree.cElementTree as ET

# import suffix.py
import suffix

OSMFILE = "data/austin.osm"

# instantiate a suffix table for use later
suffix_table = suffix.SuffixTable()

def titleize(string):
   """ capitalizes each word in a string """
   return string.title()

# construct an expected list of suffixes
expected = map(titleize, suffix_table.suffixes)

def is_word(string):
   """ returns if a string is word-like """
   return re.match(r'\w+', string) != None

def clean_suffix(string):
   """ converts a suffix abbreviation or alias into long-form suffix """
   return suffix_table.convert(string.lower()).title()

def clean_street_address(street_address):
   """ cleans up a street name """
   # split street name at word/non-word boundaries
   street_address_split = re.split(r'(\W+)', street_address)

   # iterate through indexes of split street name array
   for i in range(len(street_address_split)):
      word = street_address_split[i]

      # replace any periods in word
      if '.' in word:
         street_address_split[i] = word.replace('.', '')
```

```
        # check if word-like and is present in suffix table
        if is_word(word) and suffix_table.has_suffix(word):
            # if true, clean the suffix
            street_address_split[i] = clean_suffix(word)
    return ''.join(street_address_split)

def audit_street_type(street_types, street_name):
    # initialize suffix string to None
    suffix_string = None

    # iterate through split street name from the end
    # (most suffixes are located at the end, e.g. '3rd St.')
    for word in re.split(r'(\W+)', street_name)[::-1]:
        # check for match in suffix table, clean and set it, then break
        if suffix_table.has_suffix(word):
            suffix_string = clean_suffix(word)
            break

    # if valid suffix found, set it to the street type
    if suffix_string:
        street_type = suffix_string

    # if not, set it to the last word
    else:
        street_type = street_name.split()[-1]
        # if not in expected, add it to street types dict
        if street_type not in expected:
            street_types[street_type].add(street_name)

def audit_zipcode(invalid_zipcodes, zipcode):
    """ adds any invalid zipcodes (not in format `ddddd`) to a dict """
    if not re.match(r'^\d{5}$', zipcode):
        invalid_zipcodes[zipcode] += 1

def is_street_name(elem):
    """ returns if element is of key type street address """
    return elem.attrib['k'] == "addr:street"

def is_zipcode(elem):
    """ returns if zip-code like """
    return 'zip' in elem.attrib['k']

def audit(osmfile, limit=-1):
    """ runs audit routine, up to a custom limit (default: none) """
    # open osm file
    osm_file = open(osmfile, "r")

    # initialize data storage
```

```
    street_types = defaultdict(set)
    invalid_zipcodes = defaultdict(int)

    # initialize counter to zero
    counter = 0

    # use iterparse to go through elements
    for _, elem in ET.iterparse(osm_file, events=("start",)):
        # check if node or way type
        if elem.tag == "node" or elem.tag == "way":
            # iterate through `tag` children
            for tag in elem.iter("tag"):
                # if street name, audit as street
                if is_street_name(tag):
                    audit_street_type(street_types, tag.attrib['v'])
                # if zipcode, audit as zipcode
                if is_zipcode(tag):
                    audit_zipcode(invalid_zipcodes, tag.attrib['v'])

        # interim update every 100k records
        if counter % 100000 == 0:
            print "Processing tag #{}...".format(counter)

        # break if exceed limit
        if limit > 0 and counter > limit:
            break

        # increment loop counter
        counter += 1

    # return data
    return street_types, invalid_zipcodes

def test_audit(limit=10**7):
    # call audit function
    st_types, zipcodes = audit(OSMFILE, limit)

    # convert data to standard dicts
    street_data = dict(st_types)
    zipcode_data = dict(zipcodes)

    # fix to JSON-read/writeable
    for key in street_data:
        street_data[key] = list(street_data[key])

    # print data
    pprint(street_data)
    pprint(zipcode_data)
```

```
    # dump data to JSON files
    json.dump(street_data, open(OSMFILE + '-street-audit.json', 'w'))
    json.dump(zipcode_data, open(OSMFILE + '-zipcode-audit.json', 'w'))

    # return data
    return street_data, zipcode_data

if __name__ == '__main__':
    test_audit(limit=-1)
```