Abstract:
I recoded my previous script to use sqlite3. The main changes I made were:

- Storing bill introduction date in a table for easy before/after determination.
- Storing all the noun phrases in a flat table for easy retrieval.
- Using a variety of SQL `count()` and `GROUP BY` clauses instead of code.
- Storing the completed tfidf score in a table of it's own.

These changes allow me to make incremental additions, deletions, or alterations to the data.

I also added parameter interpretation so that you can invoke the script in either 'load' or 'display' mode. This way you don't have to wait for the data to reload if you just want to see the results again.

Data set:
My data set remains the plain text of all congressional bills enrolled (passed) for the 10th, 105th, and 106th congresses.

Method:
It required only a slight tweak to my code to use the Python standard sqlite3 module for data persistence.

I did re-factor the code somewhat to use SQL constructs like 'GROUP BY' and 'COUNT' to save some lines of Python code.

This was done to follow the pragma - 'keep the processing near the data'... conceptually at least. Since the db is hosted on the same machine as the Python script, there is likely any real benefit of doing this. If the script were ported to using a client/server model of db access, the gains would be more apparent.

Code:

```python
#!/usr/bin/python

import glob
import math
import os
import pprint
import re
import sqlite3
import sys
import time

sys.path.insert(0, '/home/mpsimmon/Desktop/montylingua-2.1/python')

import MontyLingua

def main(ARGV = sys.argv):
    dbconn = sqlite3.connect('/tmp/arbitrary-file-name')
    cursor = dbconn.cursor()
    if ARGV[0] == 'load':
        cursor.execute(''
        drop table if exists billnp
        '')
        cursor.execute(''
        drop table if exists billdate
        ')
        cursor.execute(''
    dbconn.close()
```

drop table if exists billtfidf

```sql
cursor.execute('''
create table billnp
    (billno text, billnp text, billyear date)
''')
cursor.execute('''
create table billdate
    (billno text, introdate date)
''')
cursor.execute('''
create table billtfidf
    (billno text, np text, tfidf real)
''')
```

```python
os.chdir('/foo/montylingua-2.1/python')
ml = MontyLingua.MontyLingua()
pp = pprint.PrettyPrinter(indent=2)
sample_years = ('106th', '107th', '108th')
for cong in sample_years:
    bill_path = '/home/mpsimmon/fedcode/%s/enr' % cong
    bill_files = glob.glob(bill_path + '/*.txt')
    (introdates, valfiles) = GetDates(bill_files, cong)
    for daterow in introdates.iteritems():
        cursor.execute('insert into billdate values (?,?)',
                       daterow)
        dbconn.commit()

    #get the noun phrases from each of these Bills.
    #list of all noun phrases in each file keyed by billno.
    bill_np = GetNounPhrases(valfiles, ml)
    for billno in bill_np:
        billyear = introdates[billno][:4]
        for np in bill_np[billno]:
            cursor.execute('insert into billnp values (?,?,?)',
                            (billno, np, billyear))
            dbconn.commit()

    idfrows = cursor.execute('''
select billnp, count(distinct(billno, billyear))
from billnp
group by billnp
''').fetchall()
    idf = {}
    for (np, count) in idfrows.iteritems():
        idf[np] = math.log(len(valfiles)/count)
        tf_before = {}
        tf_after = {}
        divisor = time.strptime('09-11-2001', '%m-%d-%Y')
        for billno in introdates.keys():
            tfdict = tf_after
            if time.strptime(introdates[billno], '%m-%d-%Y') <
            tfdict = tf_before
            allnp_count = cursor.execute('''
select count(*)
from billnp
where billno = ?''', billno).fetchone()[0]
```
tfrows = cursor.execute(""
    select np, count(*)
    from billnp
    where billno = '?'
    group by np""); billno).fetchall()
for row in tfrows:
    np = tfdict[billno][row[0]]
    tf = float(allnp_count/row[1])
    tfidf = tf*idf[np]
    cursor.execute('insert into billtfidf values(?,?,?)',
        (billno, np, tfidf))
    dbconn.commit()
elif ARGV[0] == 'display':
    afterrows = cursor.execute(''
        select bd.billno, bt.np, bt.tfidf
        from billdate bd, billtfidf bt
        where bd.introdate > date("2001-09-11")
        and bd.billno = bt.billno
        order by bt.tfidf desc
        limit 100''').fetchall()
    beforerows = cursor.execute(''
        select bd.billno, bt.np, bt.tfidf
        from billdate bd, billtfidf bt
        where bd.introdate < date("2001-09-11")
        and bd.billno = bt.billno
        order by bt.tfidf desc
        limit 100''').fetchall()
    print "Noun phrases from before 0911"
    for row in beforerows:
        print " %s	%s	%s" % row
    print
    print "Noun phrases from after 0911"
    for row in afterrows:
        print " %s	%s	%s" % row
    print
else:
    print 'invalid operation. valid operations are: "load" and
    "display"

def GetDates(bill_files, cong):
    '''Return the introduction and passage date of the bill. '''
    base_path = '/home/mpsimmon/fedcode/%s/' % cong
    #all the bills passed introduced before 09/11/2001
    before = []
    #all the bills passed introduced after 09/11/2001
    after = []
    date_re = re.compile(r'''
        \((\s+\w+\s+)\s+(\d{1,2})\s+(\d{4})\)''', re.X)
    s_search_order = ('is', 'ats', 'rfh')
    h_search_order = ('ih', 'rds', 'rfs', 'ath')
    billdates = []
    valfiles = []
    for file_path in bill_files:
        billno = file_path.split('/')[-1]
billbase = billno[:-7]
if billno[0] == 's':
    so = s_search_order
else:
    so = h_search_order
filex = False
for suffix in so:
    if os.path.isfile(base_path+billbase+suffix+'.txt') and not filex:
        filex = True
        f_path = base_path+billbase+suffix+'.txt'
        match = date_re.search(''.join(open(f_path).readlines()))
        if match == None:
            print billno
            continue
        idate = match.group(0)
        dbdate = time.strftime('%Y-%m-%d', time.strptime(idate,
        '%B %d, %Y'))
        billdates[billbase] = dbdate
        valfiles.append(file_path)
return (billdates, valfiles)

def GetNounPhrases(bill_list, ml):
    '''Get the monty lingua dict and return a {billno: [np, np, ...
    ...]}'''
    output = {}
    for bill in bill_list:
        billno = bill.split('/')[-1]
        b = billno[:-7]
        output[b] = []
        for section in ml.jist(''.join(open(bill).readlines())):
            output[b] += [x.lower() for x in section['noun_phrases']]
if len(x) > 2]
    return output

if __name__ == '__main__':
    main()

Output:
This change in code did not affect the outputted data. It remains the same as homework 1, and is thus omitted.

Conclusion:
It makes a great deal of sense to abstract the data pre-processing and storage away from the manipulation and display. The data set should only need to be build once, regardless of the iterations in the manipulation and display methodologies. Since the computation and extraction is the lengthliest part, this results in much greater time efficiency.