2012 USGS National Groundwater Workshop Using `python` to Improve Groundwater Model Effectiveness: NWIS Example Code

August 9, 2012

1 Getting started

This document describes code to pull groundwater level data from NWIS, parse the results, and make simple plots. The objectives of this code are to illustrate a variety of ways to read data from text files, a simple connection to an online database, and plotting the results in an external file. Along the way of developing this code, several tricks are encountered that we hope will be transferable to a variety of situations.

The remainder of this document shows the code, listed by filename and provides a brief description of the block of code presented with emphasis on tricks.

The intention of this code is to demonstrate concepts from the course, but some assumptions about the data are made and may require further modification to fit specific situations by users, so please do not use this code for important work without QC of the results.

The results of running this code are a text file with all the data requested from NWIS and a series of plots—one per well—showing the hydrograph with approved and provisional data identified. Figure 1 shows an example hydrograph.

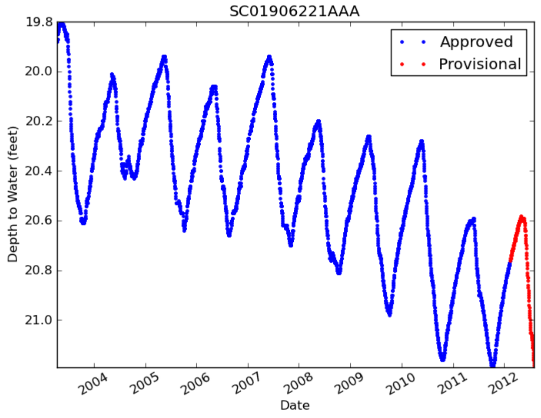

Figure 1: Example hydrograph output

2 NWIS_read_driver.py – a driver to run the entire codebase

This file loads functions from the various modules required to run the entire NWIS query example code. It is often beneficial to have a summary "driver" code like this to be brief and high level with the details contained in associated modules.

2.1 Header

The header section imports modules into appropriate namespaces and sets a single user variable called `testCase`. If `testCase=1` the code runs through the reading of the two FIPS lookup functions and returns the FIPS codes for Dane County in Wisconsin and prints results to the screen. If `testCase=2`, the entire code is run including a dialogue to request information from the user.

```
import numpy as np
import NWIS_read_parse as NWR
import NWIS_web_puller as NWISWebPull

testCase = 2
```

2.2 testCase=1

```
if testCase == 1:
    #
    # JUST RUN THROUGH THE NWIS_read_parse functions
    #

    # get the FIPS codes lookups for states and counties
    state_lookup_full,state_lookup_abbrev,county_lookup,state_abbrev_full = NWR.read_state_county_FIPS()

    # now find the specific code for place of interst
    cState = 'WI'
    cCounty = 'Dane County'
    sFIPS,cFIPS = NWR.get_county_and_state_FIPS(cState,cCounty,state_lookup_full,
                                                state_lookup_abbrev,county_lookup)

    print 'FIPS code for %s is %s\nFIPS code for %s is %s' %(cState,sFIPS,cCounty,cFIPS)
```

2.3 testCase=2

```
if testCase == 2:
    #
    # RUN THROUGH THE DIALOGUE OF PULLING RECORDS AND USE THE OUTPUT FILE TO PARSE AND PLOT
    #

    infile = NWISWebPull.Web_pull_driver()
    # example using a static file already in place

    cdat,station_lookup = NWR.NWIS_reader(infile)

    NWR.NWIS_plotter(cdat,station_lookup,'.png')

    print " All done! Check the 'figures' subdirectory for your plotted results"
```

3 NWIS_read_parse.py – a set of readers to open and parse files

This code contains functions to read several files necessary to perform the NWIS query and to interpret the results obtained by such a query.

3.1 Snippet of an output file from NWIS query

In the file listing below, note that comments are generally identified by the "#" symbol. However, we need the information about sites found in the line

`# Data for the following 1 site(s) are contained in this file`

After that, the data lines are identified by “`USGS`” at the beginning of the line.

```
# ---------------------------------- WARNING ----------------------------------------
# Provisional data are subject to revision. Go to
# http://waterdata.usgs.gov/nwis/help/?provisional for more information.
#
# File-format description:  http://waterdata.usgs.gov/nwis/?tab_delimited_format_info
# Automated-retrieval info: http://waterdata.usgs.gov/nwis/?automated_retrieval_info
#
# Contact:   gs-w_support_nwisweb@usgs.gov
# retrieved: 2012-07-24 12:57:18 EDT       (caas01)
#
# Data for the following 1 site(s) are contained in this file
#    USGS 431312089475301 DN-09/06E/29-0083
# -----------------------------------------------------------------------------------
#
# Data provided for site 431312089475301
#    DD parameter statistic   Description
#    01   72019     00001     Depth to water level, feet below land surface (Maximum)
#
# Data-value qualification codes included in this output:
#     A  Approved for publication -- Processing and review completed.
#     P  Provisional data subject to revision.
#
agency_cd        site_no        datetime        01_72019_00001        01_72019_00001_cd
5s        15s        20d        14n        10s
USGS        431312089475301        1987-01-01        9.82        A
USGS        431312089475301        1987-01-02        9.84        A
```

3.2 Header

```
import numpy as np
from datetime import datetime
import matplotlib as mpl
mpl.rcParams['pdf.fonttype'] = 42 # this strange bit of code makes fonts
                                  # editable in Adobe Illustator
import matplotlib.pyplot as plt
import re                         # regular experessions module
import os                         # operating system module
```

3.3 NWIS_reader

In this example, we explore the reading and parsing of a text file that has irregular structure. A sidenote is the use of the `datetime` module for the interpretation and conversion of date and time information. A link to further documentation is included in the comments.

In the main loop over the input lines, the main logic is to determine whether the key line identifying the number of stations reported is encountered in the comments, or whether a data line is encountered, as identified by ‘`USGS`” at the start of the line. Each line is parsed and then data placed into lists. The lists are later converted into `numpy` arrays. Note that all string comparisons are made by forcing comparison strings to lower case to avoid misfits based only on text case.

At the end of the code, the results are put into a dictionary to be returned. We use `np.nonzero` to find indices of an array meeting a specific condition. `np.where` is another way to do the same.

```
#--Function to read a file from NWIS query output
def NWIS_reader(infile):
    '''
    NWIS_reader(infile)
    A function to read in an NWIS file generated using USGS webservices.
    Mike Fienen - 7/16/2012
    <mnfienen *at* usgs *dot* gov>

    INPUT:
    infile --> the name of an input file in USGS RDB (tab-delimited) format

    OUTPUT:
    indat --> a dictionary with keys corresponding to site numbers and each
            element being a dictionary with keys date and depth to water.
    '''
    # tell the user what's happening
    print 'Reading Data from file: %s' %(infile)
    #
    # set up a couple initial variables
    #
    # format for reading the date --> formats noted at
    #        http://docs.python.org/library/datetime.html (bottom of the page)
    indatefmt = "%Y-%m-%d"


    # open the text file and read all lines into a variable "tmpdat"
    tmpdat = open(infile,'r').readlines()

    # make empty lists to temporarily hold the data
    Site_ID = []   # site ID
    dates = []     # date of measurement (daily values)
    DTW = []       # depth to water below land surface (feet)
    prov_code = [] # provisional code: [P] is provisional, [A] is accepted

    # loop over the input data, keep only proper data rows. Parse and assign to lists
    for lnum, line in enumerate(tmpdat):
        # first read the lookup information from the header of the file
        if ("data for the following" in line.lower()):
            nWells = int(re.findall("[0-9]+",line)[0])
            statnums = []
            countynums = []
            for cwell in np.arange(nWells):
                nextline = lnum+1+cwell
                tmp = tmpdat[nextline].strip().split()
                statnums.append(tmp[2])
                countynums.append(tmp[3])
            station_lookup = dict(zip(statnums,countynums))

        if (('usgs' in line.lower()) and ('#' not in line)):
            tmp = line.strip().split() # strip newline off the end and split on whitespace
            Site_ID.append(tmp[1])
            dates.append(datetime.strptime(tmp[2],indatefmt)) #convert date to a time tuple
            DTW.append(tmp[3])
            prov_code.append(tmp[4].lower()) # --> note conversion to lower case!


    Site_ID = np.array(Site_ID,dtype=str) # convert to a numpy array
                                          # keep as str to not lose trailing zeroes (!)
    DTW = np.array(DTW,dtype=float) # convert to numpy array as a float containing the measurements
    dates = np.array(dates,dtype=object) #convert to numpy array as object since these are time tuples
    prov_code = np.array(prov_code,dtype=str) # convert to a numpy array as string
    unique_sites = np.unique(Site_ID) # get a unique list of the sites

    # make an empty dictionary to include the results
    indat = dict()
    # now loop over the unique sites and parse the results by site number
    for csite in unique_sites:
        cindices = np.nonzero(Site_ID == csite)[0] # find the indices for the current site
        indat[csite] = {'dates':dates[cindices],
                        'DTW':DTW[cindices],
                        'prov_code':prov_code[cindices]}
    # tell the user we are done!
    print "Reading %s complete!" %(infile)
    return indat,station_lookup
```

3.4 read_state_county_FIPS

In order to query NWIS using a URL (discussed later), we need FIPS codes (Federal Information Processing Standard) to represent state and county identification. We provide lookup files for both FIPS codes and read them in from files in two different formats requiring two levels of parsing. The county codes are more complicated to read and are performed first. The state codes are read in using `np.genfromtxt` in its most general way.

Note the use of `np.hstack`, `np.reshape`, and `.T` (the transpose operator) to manipulate a concatenation of three arrays into one for use in lookups.

The dictionaries returned provide the ability to lookup FIPS code by state either using the full state name or the abbreviated state name. Another dictionary allows for finding the abbreviation of a state name using the full name. Finally, `county_lookup` is a 3-column `numpy` array containing state code, county code, and county name.

```
#--Function to make a lookup of State and County FIPS codes
def read_state_county_FIPS():
    '''
    read_state_county_FIPS()
    A function to read FIPS files for lookup and return two kinds of output.
    Mike Fienen - 7/16/2012
    <mnfienen *at* usgs *dot* gov>

    INPUT:
    none --> the variables are all already named
    OUTPUT:
    state_lookup_full -->  dictionary with keys of full state names
                    and elements of state codes
    state_lookup_abbrev --> dictionary with keys of state name
                    abbreviations and elements of state codes
    county_lookup --> a 3 column numpy array of strings with county codes,
                    state codes, and county names
    '''
    # filenames for the State and County lookups
    state_file = os.path.join('..','NWIS_meta_data','STATE_FIPS.csv')
    county_file = os.path.join('..','NWIS_meta_data','COUNTY_FIPS.csv')
    #
    # read in the county file --> READ THE "HARD WAY"
    #
    # make lists to hold the data we will read
    state_codes = []
    county_codes = []
    county_names = []

    indat = open(county_file,'r').readlines()  # read the whole file into memory as a list
    headers = indat.pop(0)                     # remove the 0th line and put into headers
    for line in indat:                         # go through the remaining file line by line
        tmp = line.strip().split(',')          # remove newline chars and split on ','
        state_codes.append(tmp[1])             # leave as strings to keep zero padding
        county_codes.append(tmp[2])
        county_names.append(tmp[3].lower())    # leave the names as strings but !make lower case!
    # convert the lists to numpy arrays--usefule later to enable using np.nonzero
    state_codes = np.atleast_1d(state_codes)
    county_codes = np.atleast_1d(county_codes)
    county_names = np.atleast_1d(county_names)

    # now mash them up into one numpy array
    county_lookup = np.hstack((state_codes,county_codes,county_names))
    # reshape it to be columns again and transpose the results (.T)
    county_lookup = county_lookup.reshape(3,len(county_lookup)/3).T

    #
    # read in the state file --> USES NP.GENFROMTXT
    #
    state_vals = np.genfromtxt(state_file,dtype=None,names=True,delimiter=',')
    # force the state names and abbreviations to lower case for later comparisons
    fullnames = state_vals['State_Name']
    for i in np.arange(len(fullnames)):
        fullnames[i] = fullnames[i].lower()
    abbrevnames = state_vals['State_Abbreviation']
    for i in np.arange(len(abbrevnames)):
        abbrevnames[i] = abbrevnames[i].lower()
    # now make the output lookup dictionaries to return
    state_lookup_full = dict(zip(fullnames,
                             state_vals['FIPS_Code']))
    state_lookup_abbrev = dict(zip(abbrevnames,
                               state_vals['FIPS_Code']))
    state_abbrev_full = dict(zip(fullnames,abbrevnames))
    return state_lookup_full,state_lookup_abbrev,county_lookup,state_abbrev_full
```

3.5 get_county_and_state_FIPS

Once we have made the lookup arrays and dictionaries from reading in the FIPS code files, we must link the FIPS codes to county and state names. This function allows for flexible interpretation of how the user enters county and state names and returns the codes. Note the use of `zfill` as an alternative way to pad a number with zeros. Note also the use of a `try` and `except` block for error trapping. This is not fully explained here, but in general is good practice to handle exceptions rather than relying solely on the interpreter to do so.

```
#--Function to lookup state and county FIPS codes and associate them with state and county
def get_county_and_state_FIPS(cState,cCounty,state_lookup_full,
                              state_lookup_abbrev,county_lookup):
    '''
    get_county_and_state_FIPS(cState,cCounty,state_lookup_full,
                              state_lookup_abbrev,county_lookup)
    A function to lookup a state and county set of FIPS codes using
    string representations of the states and counties, or optionally
    a FIPS for state and string for county.
    N.B.--> some of the counties are actually, parishes, etc., so some further
                    QC may be in order!
    Mike Fienen - 7/16/2012
    <mnfienen *at* usgs *dot* gov>

    INPUT:
    cState --> state to lookup, can be FIPS code, full name, or 2 letter abbrev.
    cCounty --> County name (due to note above, must include descriptor such as "county" etc.
    state_lookup_full -->  dictionary with keys of full state names
                    and elements of state codes
    state_lookup_abbrev --> dictionary with keys of state name
                    abbreviations and elements of state codes
    county_lookup --> a 3 column numpy array of strings with county codes,
                      state codes, and county names
    OUTPUT:
    cFIPS --> Dictionary with keys of "state" and "county" and elements with the FIPS codes
    '''
    #
    # first be flexible about how to handle the state code ambiguity
    #
    try:  # is it in integer?
        state_FIPS = int(cState)
    except: # if not, see, by length, if it's an abbreviation or full name
        cState = cState.lower()  # force to lower case
        lenstate = len(cState)
        if lenstate == 2:
            try:
                state_FIPS = state_lookup_abbrev[cState]
            except KeyError:
                raise KeyError('State not found')
        else:
            try:
                state_FIPS = state_lookup_full[cState]
            except KeyError:
                raise KeyError( 'State not found')
    state_FIPS = str(state_FIPS).zfill(2) # convert back to a string padded with zeros on the left
    #
    # next, find a match in county_lookup for both state FIPS code and county name
    # returning the county FIPS code
    #
    try: # is it already an integer?
        county_FIPS = int(cCounty)
    except: # if not, use the string to look up
        cCounty = cCounty.lower()  # first force to lower case for the comparison
        # now, find the indices within county_lookup that match both county name and state FIPS code
        indies = (np.where(county_lookup[:,0]==state_FIPS) and np.where(county_lookup[:,2]==cCounty))[0]
        if len(indies)>1:
            print "ambiguous county and state match"
        else:
            county_FIPS = county_lookup[indies,1][0]
    county_FIPS=str(county_FIPS).zfill(3) # convert back to string and pad with zeros on left
    return state_FIPS,county_FIPS
```

3.6 NWIS_plotter

The final step for our code is to plot results of hydrographs using `matplotlib`. The user provides the data returned by the `NWIS_reader` function. The user also specifies the output format for the file to be saved (e.g. .png or .pdf) and a flag determining whether the plots should be displayed to the screen in addition to being saved to disk. In the function call, the use of a default value (`disp_plot=False`) makes a variable optional. If the variable is left out of the function call, the default value is used.

Note also the first if statement regarding the subdirectory called figures. This is a common trick to direct output to a subdirectory without requiring the directory to be present already.

The plotting is straightforward, although note the handling of provisional and approved data.

```
#--Function to plot hydrographs from an NWIS query using MATPLOTLIB
def NWIS_plotter(indat,station_lookup,plot_format,disp_plot=False):
    '''
    NWIS_plotter(indat,station_lookup,plot_format,disp_plot)

    A function to plot each hydrograph from an NWIS data query. The alternate name is used as the title
    Mike Fienen - 7/16/2012
    <mnfienen *at* usgs *dot* gov>

    INPUT:
    indat --> a dictionary of data returned from NWIS_reader
    station_lookup --> a dictionary linking station ID to station name
    plot_format --> format to save files of plots. can be '.png' or '.pdf'
    disp_plot --> a Boolean flag determining whether or not to display the plots onscreen

    OUTPUT:
    a set of files with plots of depth to water (reversing y axis) and named with station ID
    '''
    # check to see if there's already a figures subdirectory. If not, make one!
    if os.path.exists('figures')==False:
        os.mkdir('figures')

    for cstation in station_lookup.keys():
        # tell the user what's going on
        print 'plotting Station ID: %s' %(cstation)
        #
        # first parse all the data for the current station
        #
        cname = station_lookup[cstation] # find the name from the stationID lookup dictionary
        DTW = indat[cstation]['DTW'] # get the depth to water for the current station
        prov_code = indat[cstation]['prov_code'] # get the provisional/approved codes
        dates = indat[cstation]['dates'] # pull the dates as datetime objects
        p_inds = np.nonzero(prov_code=='p')[0]
        a_inds = np.nonzero(prov_code=='a')[0]

        #
        # then set up the figure and plot it
        #
        fig = plt.figure() # make a figure
        ax = fig.add_subplot(111)  # make a handle (ax) to the axes object in the figure
        plt.hold = True  # hold the figure to allow multiple plotting events
        if len(a_inds) > 0:
            plt.plot(dates[a_inds],DTW[a_inds],'b.')
        if len(p_inds) > 0:
            plt.plot(dates[p_inds],DTW[p_inds],'r.')
        ax.set_ylim([np.max(DTW),np.min(DTW)])

        # set the title and label the axes
        plt.title(cname)
        plt.ylabel('Depth to Water (feet)')
        plt.xlabel('Date')
        # rotate the xtick labels to avoid dates overlapping
        plt.xticks(rotation = 30)
        # finally, add a legend
        if len(p_inds) > 0 and len(a_inds) > 0:
            plt.legend(['Approved','Provisional'],loc='best')
        elif len(p_inds) > 0:
            plt.legend(['Provisional'],loc='best')
        elif len(a_inds) > 0:
            plt.legend(['Approved'],loc='best')
        plt.gcf().subplots_adjust(bottom=0.15) # make a little room for the date labels

        plt.savefig(os.path.join('figures',cstation + plot_format))
    if disp_plot:
        plt.show()
```

4 NWIS_web_puller.py – a module to construct and retrieve NWIS URLs

This code constructs the queries necessary to pull NWIS data from a USGS webservice using RESTful queries. REST stands for Representational State Transfer—what this really means is that a user can create a URL with a few variables defined in line to retrieve data through a web browser. The `urllib` built-in `python` module also provides a means to get data from a URL.

4.1 Header

The only new module being loaded here relative to the other codes is `urllib`

```
import numpy as np
from matplotlib import pyplot as plt
import matplotlib as mpl
mpl.rcParams['pdf.fonttype'] = 42
import os
import NWIS_read_parse as NWR
import urllib
```

4.2 Web_pull_driver – a driver to obtain user information and run the other web pulling functions

This driver code uses `raw_input` to obtain options from the user to obtain NWIS data. Examples of running the code are as follows:

```
>>> import NWIS_web_puller as NWP
>>> NWP.Web_pull_driver()
You are about to enter the NWIS puller zone
Ready?


Would you like to query by Station Numbers [station]
State [state], or County and State [county]?:state
Please enter state as full, abbreviated, or FIPS code:Co
Please enter start date in format "YYYY-MM-DD":1900-01-01
Please enter end date in format "YYYY-MM-DD":2012-08-01
Please enter a filename for your results:COtest.dat
```

```
Pulling data for Co
Using URL:
http://waterservices.usgs.gov/nwis/dv/?format=rdb&stateCd=Co&startDT=1900-01-01&endDT=2012-08-01&parameterCd=72019&siteType=GW
File download complete
output written to COtest.dat
```

```
>>> import NWIS_web_puller as NWP
>>> NWP.Web_pull_driver()
You are about to enter the NWIS puller zone
Ready?


Would you like to query by Station Numbers [station]
State [state], or County and State [county]?:station
Please enter 15-digit station codes, separated by commas:372106105241701,372141105281101
Please enter start date in format "YYYY-MM-DD":1900-01-01
Please enter end date in format "YYYY-MM-DD":2012-08-01
Please enter a filename for your results:co_some.dat
Pulling data for list of stations
Using URL:
http://waterservices.usgs.gov/nwis/dv/?format=rdb,1.0&sites=372106105241701,372141105281101&startDT=1900-01-01&endDT=2012-08-01&parameterCd=72019&siteType=GW
File download complete
output written to co_some.dat
```

The code listing is as follows. Note that all input from `raw_input` is a string so conversions are required if numerical or other information is required.

```
# ######## #
 #  M A I N #
  # ######## #

#--A driver that asks the user about which data they want to obtain and runs all the functions to get it
def Web_pull_driver():
    '''
    NWIS_web_puller()
    A coad to download an NWIS file using USGS webservices.
    Mike Fienen - 7/16/2012
    <mnfienen *at* usgs *dot* gov>

    INPUT:
    none--> user is prompted for all input
    OUTPUT:
    none--> a text file is written by the USGS weservices with a name
    specified by the user
    '''
    print 'You are about to enter the NWIS puller zone\nReady?\n\n'

    stat_county = raw_input('Would you like to query by Station Numbers [station] \nState [state], or County and State [county]?:')

    if stat_county.lower() == 'county' or stat_county.lower() == 'state':
        cState = raw_input('Please enter state as full, abbreviated, or FIPS code:')
        if stat_county.lower() == 'county':
            cCounty = raw_input('Please enter county name or FIPS code\ninclude "county", "parish", etc:')
        else:
            cCounty = '999999'
        sttime = raw_input('Please enter start date in format "YYYY-MM-DD":')
        endtime = raw_input('Please enter end date in format "YYYY-MM-DD":')
        outfilename = raw_input('Please enter a filename for your results:')
        retrieve_by_state_county(cState,cCounty,sttime,endtime,outfilename,stat_county.lower())
    elif stat_county.lower() == 'station':
        tmp_input = raw_input('Please enter 15-digit station codes, separated by commas:')
        sttime = raw_input('Please enter start date in format "YYYY-MM-DD":')
        endtime = raw_input('Please enter end date in format "YYYY-MM-DD":')
        outfilename = raw_input('Please enter a filename for your results:')
        tmp = tmp_input.split(',')
        cStations = []
        for i in tmp:
            cStations.append(i)
        retrieve_by_stations(cStations,sttime,endtime,outfilename)
    else:
        print 'Invalid station, state, or county selection'
    return outfilename
```

4.3 `retrieve_by_state_county`

One option is to pull data by specifying a state and county (or by state only). The behavior is controlled by the variable `stat_county`. A list called `urlParts` contains the various elements of the URL for the RESTful query and the county and state information are placed in the appropriate locations to create a full URL. Pulling the data is as simple as reading the data stream into memory and then writing the results to a file using binary protocol. The binary protocol (signified by `'wb'` in the output file open statement) works for binary or text files so is the most general solution. It is not really necessary to save to a file—one could leave the data stream in memory instead, but it's often useful to save an archive of exactly what was pulled in the query.

```
#--Function to query NWIS for all the wells in a State County combination
def retrieve_by_state_county(cState,cCounty,sttime,endtime,outfilename,stat_county):
    if stat_county == 'county':
        print 'Pulling data for %s in %s' %(cCounty, cState)
    elif stat_county == 'state':
        print 'Pulling data for %s' %(cState)
    state_lookup_full,state_lookup_abbrev,county_lookup,state_full_abbrev_lookup = NWR.read_state_county_FIPS()
    stateFIPS,countyFIPS = NWR.get_county_and_state_FIPS(cState,cCounty,state_lookup_full,state_lookup_abbrev,county_lookup)
    urlParts = ['http://waterservices.usgs.gov/nwis/dv/?format=rdb',
            '&countyCd=',
            '&stateCd=',
            '&startDT=',
            '&endDT=',
            '&parameterCd=72019&siteType=GW']
    if stat_county == 'county':
        fullURL = urlParts[0] + urlParts[1] + stateFIPS + countyFIPS + urlParts[3] + sttime + urlParts[4] + endtime + urlParts[5]
    elif stat_county == 'state':
        if len(cState) > 2:
            cState = state_lookup_abbrev[cState]
        fullURL = urlParts[0] + urlParts[2] + cState + urlParts[3] + sttime + urlParts[4] + endtime + urlParts[5]

    print 'Using URL:\n%s' %(fullURL)
    datastream = urllib.urlopen(fullURL).read()
    open(outfilename,'wb').write(datastream)
    print 'File download complete'
    print 'output written to %s' %(outfilename)
```

4.4 retrieve_by_stations

The other option is to pull data by specifying a a set of station IDs, separated by commas. This function works almost the same as `retrieve_by_state_county` except that the URL is constructed using station IDs.

```
#--Function to query NWIS with a list of station numbers
def retrieve_by_stations(cStations,sttime,endtime,outfilename):
    print 'Pulling data for list of stations'
    urlParts = ['http://waterservices.usgs.gov/nwis/dv/?format=rdb,1.0&sites=',
                '&startDT=',
                '&endDT=',
                '&parameterCd=72019&siteType=GW']
    fullURL = urlParts[0]
    for i,cs in enumerate(cStations):
        fullURL += cs
        if i+1 < len(cStations):
            fullURL += ','
    fullURL += urlParts[1] + sttime + urlParts[2] + endtime + urlParts[3]
    print 'Using URL:\n%s' %(fullURL)
    datastream = urllib.urlopen(fullURL).read()
    open(outfilename,'wb').write(datastream)
    print 'File download complete'
    print 'output written to %s' %(outfilename)
```