Read Images and Clip

```
try:
   import arcpy
except ImportError:
   print "You need arcpy installed"
   sys.exit ( -1 )
from arcpy import env
env.overwriteOutput = True

import fileinput
import os
from os import listdir
from os.path import isfile, join
import csv
try:
   import numpy as np
except ImportError:
   print "You need numpy installed"
   sys.exit ( -1 )
import inspect
import logging
import pickle

# Set environment settings
env.workspace = "F:/py"
working_dir = os.getcwd()

settings = {
'Image Dump': '',
'Image List': 'Master List.csv',
'Output Dir': 'Processed',
'Clip Dir': 'Clips',
'Scene Dir': 'image dump',
# reading master list
'Code Header': 'Code',
'Scene Header': 'Scene Name',
'Pick Bands': ['1', '2', '3', '4', '5', '6', '7'],
'Parameter List': ['dnmin', 'gain', 'bias', 'lmax', 'lmin', 'qc_lmax', 'qc_lmin'],
# metadata store
'MD file':'metadata.txt'
}

# code adapted from J Gomez-Dans <j.gomez-dans@ucl.ac.uk>
def process_metadata ( fname , scene):
   """A function to extract the relelvant metadata from the
   USGS control file. Returns dicionaries with LMAX, LMIN,
   QCAL_LMIN and QCAL_LMAX for each of the bands of interest."""
```

```
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))

    # band vars
    lmax = {} # Dicts to store constants
    lmin = {}
    qc_lmax = {}
    qc_lmin = {}
    gain = {}
    bias = {}
    p_list = ['lmax', 'lmin', 'qc_lmax', 'qc_lmin', 'gain', 'bias']
    parameters = [lmax, lmin, qc_lmax, qc_lmin, gain, bias]

    # scene vars
    sun_elevation = 0
    julian_date = 0
    sat_id = ''

    # lets just go ahead and assemble these as a dict for return
    result = {}

    with open(fname, 'r') as fp: # Open metadata file
        for line in fp:
          if ( line.find ("RADIANCE_MULT_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip() + '.TIF'
            logger.debug('test {} in {}: {}'.format(the_band, settings['Pick Bands'], the_band in settings['Pick
Bands']))
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
              gain[band_name] = float ( s[-1] ) # Get constant as float
              logger.debug('gain[{}]: {}'.format(band_name, gain[band_name]))
          elif ( line.find ("RADIANCE_ADD_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[3].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
              bias[band_name] = float ( s[-1] ) # Get constant as float
          elif ( line.find ("QUANTIZE_CAL_MAX_BAND") >= 0 ):
            s = line.split("=") # Split by equal sign
            # test this one
            the_band = s[0].strip()[-1] # Band number as string
            # use this as dict name
            band_name = 'B' + s[0].split("_")[4].strip()
            if the_band in settings['Pick Bands']: # Is this one of the bands we want?
```

```
                qc_lmax[band_name] = float ( s[-1] ) # Get constant as float
            elif ( line.find ("QUANTIZE_CAL_MIN_BAND") >= 0 ):
                s = line.split("=") # Split by equal sign
                # test this one
                the_band = s[0].strip()[-1] # Band number as string
                # use this as dict name
                band_name = 'B' + s[0].split("_")[4].strip()
                if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                    qc_lmin[band_name] = float ( s[-1] ) # Get constant as float
            elif ( line.find ("RADIANCE_MAXIMUM_BAND") >= 0 ):
                s = line.split("=") # Split by equal sign
                # test this one
                the_band = s[0].strip()[-1] # Band number as string
                # use this as dict name
                band_name = 'B' + s[0].split("_")[3].strip()
                if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                    lmax[band_name] = float ( s[-1] ) # Get constant as float
            elif ( line.find ("RADIANCE_MINIMUM_BAND") >= 0 ):
                s = line.split("=") # Split by equal sign
                # test this one
                the_band = s[0].strip()[-1] # Band number as string
                # use this as dict name
                band_name = 'B' + s[0].split("_")[3].strip()
                if the_band in settings['Pick Bands']: # Is this one of the bands we want?
                    lmin[band_name] = float ( s[-1] ) # Get constant as float
            elif ( line.find('SUN_ELEVATION') >= 0):
                s = line.split('=')
                sun_elevation = s[1].strip()
                log.debug('sun_elevation: {}'.format(sun_elevation))

        # other is scene wide metadata
        sat_id = scene[2]
        julian_date = scene[13:16]
        log.debug('sat id: {}, julian_date: {}'.format(sat_id, julian_date))
        other = [sat_id, julian_date, sun_elevation]

        # make named dict for result
        for (x, i) in zip(p_list, parameters):
                result[x] = i
        log.debug('result: {}'.format(result))
        return [result, other]

    # code adapted from J Gomez-Dans <j.gomez-dans@ucl.ac.uk>
    def get_metadata ( fname ):
        """This function takes `fname`, a filename (opionally with a path), and
        and works out the associated metadata file"""
        log = logger.getChild(inspect.currentframe().f_code.co_name)
        log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
```

```
    original_fname = os.path.basename ( fname )
    metadata_fname = original_fname.split("_")[0] + "_MTL.txt"
    metadata_fname = os.path.join ( os.path.dirname ( fname ), metadata_fname )

    return metadata_fname

def discover_files(loc):
    """
    Search curdir for a csv or txt in the Raw Input dir.
    This is the list of locations and dates which are used to select images from
    the other directory full of LS metadata "Target List"
    Lets use a working dir, and then a file name "Test list.xls". load the xlreader.
    Read check for clip, read for dnmin, then clip and save to new dir.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    result = []
    # Load all Input_Types files in curdir to a list
    file_list = [ f for f in listdir(loc) if isfile(join(loc,f)) \
       and os.path.splitext(f)[1] in settings['Input_Types']]

    log.debug('found anything?: {}'.format(file_list))

    # See if it's a CSV or TXT file
    for file in file_list:
       extention =  os.path.splitext(file)[1]
       log.debug('checking extention: {}'.format(extention))
       if extention.lower() == '.csv':
          log.debug('found a csv file: {}'.format(file))
          result.append(join(loc,file))
       elif extention.lower() == '.txt':
          log.debug('found a txt file: {}'.format(file))
          result.append(file)
       else:
          log.debug('no csv or txt file was found')
    if verbose: print ('discover result: ', result)
    if result:
       log.debug('found an input')
    return result

def list_find(list, search_term):
    """
    Find if something is found in a list of lists, and return that column's data

    Returns ['R, C', 'List of stuff -header']
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.debug('Initializing {}'.format(inspect.currentframe().f_code.co_name))
```

```
    logger.debug('for {} in {}'.format(search_term, list))
    result = []

    # see if term is in the list
    term_row = [idx for idx in list if search_term in idx]
    log.debug('term row: {}'.format(term_row))
    # possible outcomes are 0 = not found, 1 = expected to find one time, >1 = problem...
    # this is a list comp, so result is a list. check length!
    if len(term_row) == 1:
        # get the index of the row term was found in
        term_row_i = [i for i, x in enumerate(list) if search_term in x][0]
        log.debug('term row_i: {}'.format(term_row_i))
        # get the column index it was found in
        term_col_i = [x for x, i in enumerate(term_row[0]) if i == search_term][0]  # returns i as list, so [0]
        log.debug('term col_i: {}'.format(term_col_i))
        # with start row and col found, find last col and return a selection
        # only get from header row +1 to last row
        # can deal with an extra, blank line result on return
        selection = [x[term_col_i] for idx, x in enumerate(list) if idx in range(term_row_i +1, len(list))] # but
this leaves blank on the end of names
        log.debug('list_find selection dump: {}'.format(selection)) # should gaurd against blanks above
header row
        result = [[term_row_i, term_col_i], selection]
    else:
        log.warning(u'Failed to find {} exactly once in list'.format(search_term))
        result = [0, 0]
    log.debug('list_find result: {}'.format(result))
    return result


def main():
    pass

if __name__ == '__main__':
    main()

    # Setup Logger
    logger = logging.getLogger(os.path.basename(__file__))
    logger.setLevel(logging.DEBUG)

    formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
    format2 = logging.Formatter('%(levelname)s - %(message)s')

    # Logging to conlsole/standard output
    ch = logging.StreamHandler()
    ch.setLevel(logging.DEBUG)
    ch.setFormatter(formatter)
    logger.addHandler(ch)
```

```
# read master list
# first is it present?
input_file = settings['Image List']
input_list = []
clip_list = []
if os.path.isfile(input_file):
    with open(input_file, 'r') as csvfile:
        logger.debug('opened: {}'.format(input_file))
        reader = csv.reader(csvfile)
        for row in reader:
            input_list.append(row)
            logger.debug('read: {}'.format(row))

# check for all scenes in image dump
    # get list of clip codes
    clip_list = list_find(input_list, settings['Code Header'])[1]
    logger.debug('code len & list: {} {}'.format(len(clip_list), clip_list))
    # reduce codes to just first half (scene code)
    clip_list = [i[:2] for i in clip_list]
    clip_list = set(clip_list)
    logger.debug('uniq code & scene list: {} {}'.format(len(clip_list), clip_list))

    # check each is present in clips
    clip_dir = os.path.join(os.getcwd(), settings['Clip Dir'])
    logger.debug('clip_dir: {}'.format(clip_dir))

    # get just the prefix, only for SHP files
    clips_in_dir = [i[:2] for i in listdir(clip_dir) if i.partition('.')[2] == 'shp']
    logger.debug('clips_in_dir 1 len & list: {} {}'.format(len(clips_in_dir), clips_in_dir))

    # see if all of clip_list can be found in clips_in_dir list
    clips_missed = []
    for i in clip_list:
        if i in clips_in_dir:
            logger.debug('clip {} found already clipped'.format(i))
        else:
            logger.warning('clip {} NOT found already clipped'.format(i))
            clips_missed.append(i)

    if clips_missed:
        logger.warning('clips NOT all found already clipped: {}'.format(clips_missed))
    else:
        logger.debug('clips all found already clipped')

    # get list of scenes
    scene_list = list_find(input_list, settings['Scene Header'])[1]
    logger.debug('scene len & list: {} {}'.format(len(scene_list), scene_list))
```

```
scene_list = set(scene_list)
logger.debug('uniq len & scene list: {} {}'.format(len(scene_list), scene_list))

# Start with each scene in the input_list
scene_dir = os.path.join(os.getcwd(), settings['Scene Dir'])
logger.debug('scene_dir: {}'.format(scene_dir))
for x in scene_list:
   logger.debug('processing scene: {}'.format(x))
   parameter_dict = {}
# read metadata to store in dict
   metadata_file = get_metadata(os.path.join(scene_dir, x))
   logger.debug('metadata name: {}'.format(metadata_file))
   if os.path.isfile(metadata_file):
      parameter_dict , scene_vars = process_metadata(metadata_file, x)
      logger.debug('metadata sneak peak: {}'.format(parameter_dict))
      logger.debug('scene metadata: {}'.format(scene_vars))

      # Get all the scenes
      scenes = [i for i in listdir(scene_dir) if x in i and i.partition('.')[2] == 'TIF' ]
      logger.debug('scenes selected from scene dir 1: {}'.format(scenes))

      # drop bands not on in the Pick List
      scenes = [i for i in scenes if i.partition('_')[2][1] in settings['Pick Bands']]
      logger.debug('reduced scenes selected from scene dir : {}'.format(scenes))

      # collect each scene's DNmin
      dnmin = {}
      for i in scenes:
         scene = os.path.join(settings['Scene Dir'], i)
         logger.debug('scene path: {}'.format(scene))
         img = arcpy.Raster(scene)
         # take of extention to match what the metadata keys look like
         this_band = i.partition('_')[2].split('.')[0]

         # check if RAT present or needs built
         test = img.hasRAT
         if test:
            logger.debug('{} has RAT'.format(i))
         else:
            logger.debug('{} has no RAT'.format(i))
            t = arcpy.BuildRasterAttributeTable_management(scene, "NONE")
            if t:
               logger.debug('{} had RAT built'.format(i))
            else:
               logger.debug('{} failed to build RAT'.format(i))

         if test or t:
            # these should be acending from lowest value
```

```
        with arcpy.da.SearchCursor(scene, ["VALUE", "COUNT"]) as rows:
            for row in rows:
                d = 0
                val = row[0]
                count = row[1]
                # find the lowest value with a count of at least 100
                if int(val) > 0 and int(count) > 100:
                    d = int(val)
                    dnmin[this_band] = d
                    logger.debug('{} is DNmin'.format(d))
                    break
                else:
                    logger.debug('{} is not DNmin'.format(val))
logger.debug('DNmin: {}'.format(dnmin))
# add dnmin to parameter dict
parameter_dict['dnmin'] = dnmin
logger.debug('parameters as dict: {}'.format(parameter_dict))

# store it by scene name
Correction_Parameters[x] = parameter_dict
Correction_Parameters[x]['sat_id'] = scene_vars[0]
Correction_Parameters[x]['julian_date'] = scene_vars[1]
Correction_Parameters[x]['sun_elevation'] = scene_vars[2]
logger.debug('assembed correction parameters for {}: {}'.format(x, Correction_Parameters))

# Get all the lakes that need clipped out of this image
process_list = [i for i in input_list if x in i]
# Make sure you have something:
if process_list:
    logger.debug('lakes to be clipped: {}'.format(process_list))
    # Process this list
    for i in process_list:
        # Set clip variables
        output_dir = os.path.join(settings['Output Dir'], i[0])
        rectangle = '#'
        clip_feature = clip_dir + '/' + i[0][:2] + '_box.shp'
        nodata_value = '0'
        clipping_geometry = 'ClippingGeometry'
        maintain_clipping_extent = 'NO_MAINTAIN_EXTENT'

        # make sure the output dir exists, if not, make
        out = os.path.join(working_dir, settings['Output Dir'], i[0])
        if not os.path.exists(out):
            os.makedirs(out)

        # Clip and output all bands of the scene
        for j in scenes:
            in_raster = os.path.join(scene_dir, j)
```

```
                    out_raster = os.path.join(working_dir, output_dir, 'c_'+j)
                    # Execute Clip
                    #Clip_management in_raster rectangle out_raster {in_template_dataset} {nodata_value}
{NONE | ClippingGeometry}
                    clip = arcpy.Clip_management(in_raster, rectangle, out_raster, \
                       clip_feature, nodata_value, clipping_geometry, maintain_clipping_extent)
                    # reclass 0 to be NoData
                    if clip:
                       logger.debug('clipped scene {} to {}'.format(j, output_dir))
                    else:
                       logger.warning('failed to clip: {}'.format(j))
             else:
                logger.warning('process_list couldnt find: {}'.format(x))

          else:
             logger.critical('metadata not found!: {}'.format(metadata_file))

       # write Correction Parameters dictionary to Output's dir
       logger.debug('writing corr parameters to {}'.format(settings['MD file']))
       with open(os.path.join(settings['Output Dir'], settings['MD file']), 'wb') as handle:
          pickle.dump(Correction_Parameters, handle)

    else:
       logger.warning(u'Missing the input file: {}'.format(settings['Image List']))

    logger.debug('finished!')
```

Image Correction

```
try:
   import arcpy
except ImportError:
   print "You need arcpy installed"
   sys.exit ( -1 )
from arcpy import env
env.overwriteOutput = True
from arcpy.sa import *

#Check out the Spatial Analyst extension
try:
   arcpy.CheckOutExtension("spatial")
except ImportError:
   print "You need access to Spatial Analyst"
   sys.exit ( -1 )

import os
from os import listdir
from os.path import isfile, join, isdir
import csv
```

```
import numpy as np
import math
import inspect
import logging
import pickle
from itertools import islice

working_dir = os.getcwd()

settings = {
'Image Dump': '',
'Image List': 'Master List.csv',
'Output Dir': 'Processed',
'Clip Dir': 'Clips',
'Scene Dir': 'image dump',
'Cloud Dir': 'cloud mask',
'Extr Dir': 'Extracts', #Extract dump
'Agg File': 'agg.dbf',   # Extract file name
'Agg Output': 'Aggregated.csv',
'SD File': 'd.csv',     # Solar distance file from NASA
# reading master list
'Code Header': 'Code',
'Scene Header': 'Scene Name',
'Pick Bands': ['1', '2', '3', '4', '5', '6', '7'],
# metadata store
'MD file':'metadata.txt',
'Clip Prefix': 'c_',
'Rad Prefix': 'r',  # Converted to TOA radiance
'Corr Prefix': 'c', # COST DOS corrected
'Temp Prefix': 't', # For B6 once it's temperature
'NDWI Prefix': 'water_', # prefix for water mask
'Blue Band': 'B1',
'Green Band': 'B2',
'Red Band': 'B3',
'NIR Band': 'B4',
'SWIR Band': 'B5',
'Thermal Band': 'B6',
}

sun_d = {}

Do_COSTDOS = 1
# switch for Testing or Production
Testing = 0
if Testing:
   log_location = 'log.txt'
   output_level = logging.DEBUG
else:
```

```
    log_location = 'log.txt'
    output_level = logging.INFO


#////////////////////////  Subfunctions //////////////////////////#

# code adapted from Steve Kochaver kochaver.python@gmail.com
def calc_radiance (LMAX, LMIN, QCALMAX, QCALMIN, QCAL, outfolder):
    """
    Calculate the TOA radiance from metadata on each band.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))

    log.debug('with {}/{} {}/{} {}/{}'.format(LMAX, LMIN, QCALMAX, QCALMIN, QCAL, outfolder))
    LMAX = float(LMAX)
    LMIN = float(LMIN)
    QCALMAX = float(QCALMAX)
    QCALMIN = float(QCALMIN)
    offset = (LMAX - LMIN)/(QCALMAX-QCALMIN)
    input_ras = Raster(QCAL)
    outname = os.path.join(outfolder, (settings['Rad Prefix'] + QCAL.split('\\')[-1]))
    log.debug('output name: {}'.format(outname))

    out_raster = (offset * (input_ras - QCALMIN)) + LMIN
    log.debug('saving output as: {}'.format(outname))
    out_raster.save(outname)

    return outname

# code adapted from Steve Kochaver kochaver.python@gmail.com
def calc_reflectance(solarDist, ESUN, solarElevation, radianceRaster, Lhaze, outfolder):
    """
    COSTDOS correction in the Lhaze parameter. Toggle Global var Do_Costdos to use 0.0.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with: {} {} {} {} {} {}'.format(solarDist, ESUN, solarElevation, radianceRaster, Lhaze,
outfolder))

    outname = os.path.join(outfolder, (settings['Corr Prefix'] + radianceRaster.split('\\')[-1]))
    #Value for solar zenith is 90 degrees minus solar elevation (angle from horizon to the center of the
sun)
    #http://landsathandbook.gsfc.nasa.gov/data_properties/prog_sect6_3.html
    solarZenith = (90.0 - float(solarElevation)) * (math.pi / 180)    #Converted from degrees to radians
    #solarZenith = math.pow(((90.0 - float(solarElevation))* (math.pi / 180)), 2)
    solarDist = float(solarDist)
    ESUN = float(ESUN)
```

```
    radiance = Raster(radianceRaster)

    #outraster = (math.pi * radiance * math.pow(solarDist, 2)) / (ESUN * math.cos(solarZenith)) *
scaleFactor
    outraster = (math.pi * (radiance - Lhaze) * math.pow(solarDist, 2)) / (ESUN * math.cos(solarZenith))
    outraster.save(outname)
    log.debug('saved: {}'.format(outname))

    return outname

# code adapted from Steve Kochaver kochaver.python@gmail.com
def get_ESUN(bandNum, SIType):
    """
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))

    SIType = SIType
    ESUN = {}
    #from NASA's Landsat7 User Handbook Table 11.3
http://landsathandbook.gsfc.nasa.gov/pdfs/Landsat7_Handbook.pdf
    #ETM+ Solar Spectral Irradiances(generated using the Thuillier solar spectrum)
    #if SIType == 'ETM+ Thuillier':
    if SIType == '7':
        ESUN = {'B1':1997,'B2':1812,'B3':1533,'B4':1039,'B5':230.8,'B7':84.90,'B8':1362}

    #from NASA's Landsat7 User Handbook Table 11.3
http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html
    #ETM+ Solar Spectral Irradiances (generated using the combined Chance-Kurucz Solar Spectrum
within MODTRAN 5)
    if SIType == 'ETM+ ChKur':
        ESUN = {'b1':1970,'b2':1842,'b3':1547,'b4':1044,'b5':225.7,'b7':82.06,'b8':1369}

    #from NASA's Landsat7 User Handbook Table 9.1
http://landsathandbook.gsfc.nasa.gov/pdfs/Landsat7_Handbook.pdf
    #from the LPS ACCA algorith to correct for cloud cover
    if SIType == 'LPS ACAA Algorithm':
        ESUN = {'b1':1969,'b2':1840,'b3':1551,'b4':1044,'b5':225.7,'b7':82.06,'b8':1368}

    #from Revised Landsat-5 TM Radiometric Calibration Procedures and Postcalibration
    #Dynamic Ranges Gyanesh Chander and Brian Markham. Nov 2003. Table II.
http://landsathandbook.gsfc.nasa.gov/pdfs/L5TMLUTIEEE2003.pdf
    #Landsat 4 ChKur
    #if SIType == 'Landsat 5 ChKur':
    if SIType == '5':
        ESUN = {'B1':1957,'B2':1825,'B3':1557,'B4':1033,'B5':214.9,'B7':80.72}

    #from Revised Landsat-5 TM Radiometric Calibration Procedures and Postcalibration
```

```
    #Dynamic Ranges Gyanesh Chander and Brian Markham. Nov 2003. Table II.
http://landsathandbook.gsfc.nasa.gov/pdfs/L5TMLUTIEEE2003.pdf
    #Landsat 4 ChKur
    if SIType == 'Landsat 4 ChKur':
        ESUN = {'b1':1957,'b2':1826,'b3':1554,'b4':1036,'b5':215,'b7':80.67}

    bandNum = str(bandNum)

    return ESUN[bandNum]

def b6_to_temp(sat_id, radiance_ras):
    """
    Converts B6 as radiance to temperature in Kelvins
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(radiance_ras))
    result = []

    name = radiance_ras.rsplit('\\', 1)
    desc=arcpy.Describe(radiance_ras)
    print 'no data =', desc.noDataValue, type(desc.noDataValue)

    input_ras = Raster(radiance_ras, noDataValue=desc.noDataValue)
    outname = os.path.join(name[0], (settings['Temp Prefix'] + name[1]))
    log.debug('output name: {}'.format(outname))
    #from http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html 11.5
    calibration_constants = {
    '7': [666.09, 1282.71],
    '5': [607.76, 1260.56]
    }

    # from http://landsathandbook.gsfc.nasa.gov/data_prod/prog_sect11_3.html
    #T = (k2)/(ln(k1/L +1))
    #Where:
    #T     =   Effective at-satellite temperature in Kelvin
    #K2    =   Calibration constant 2 from Table 11.5
    #K1    =   Calibration constant 1 from Table 11.5
    #L     =   Spectral radiance in watts/(meter squared * ster * ????m)

    d1 = (calibration_constants[sat_id][0] / input_ras) +1
    out_raster = calibration_constants[sat_id][1] / Ln(d1)

    log.debug('saving output as: {}'.format(outname))
    out_raster.save(outname)

    return outname
```

```
def utm_window(args):
    """
    Accepts pair of UTM coords (list).
    Return the surrounding 8 coords in a 3x3 pixel window  plus the given "center"
    As list of list objects
    """
    center = args
    result = []

    for (x, y) in center:
        result.append([x -30, y -30])
        result.append([x -30, y -0])
        result.append([x -30, y +30])
        result.append([x -0, y -30])
        result.append([x -0, y -0])
        result.append([x -0, y +30])
        result.append([x +30, y -30])
        result.append([x +30, y -0])
        result.append([x +30, y +30])

    return result

def make_pointfile(points, out_name):
    """
    Accept a list of points and a name.
    Make a shapefile containing those points named name, return the name.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(points))
    result =[]

    pt = arcpy.Point()
    ptGeoms = []
    for p in points:
        pt.X = p[0]
        pt.Y = p[1]
        pg = arcpy.PointGeometry(pt)
        ptGeoms.append(pg)
    final_name = out_name + ".shp"
    print final_name
    arcpy.CopyFeatures_management(ptGeoms, final_name)
    del ptGeoms

    return final_name


def ndwi_mask(folder_path, B2, B5):
```

```
    """
    Accepts the path and two required bands as inputs (2 & 5)
    Creates ndwi_Imagename.TIF in same dir
    Returns status code
    Water = 1 (True), otherwise false (0)
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('in {} with {} and {}'.format(folder_path, B2, B5))
    result = 0
    output_name = os.path.join(folder_path, settings['NDWI Prefix'] + B2.rsplit('_', 1)[0] + '.TIF')

    Green = os.path.join(folder_path, B2)
    NIR = os.path.join(folder_path, B5)

    #Create Numerator and Denominator rasters as variables and NDVI output (note that arcpy.sa.Float
returns a floating point raster)
    numerator = arcpy.sa.Float(Raster(Green) - Raster(NIR))
    denominator = arcpy.sa.Float(Raster(Green) + Raster(NIR))
    log.debug("Dividing")
    NDWI_eq = arcpy.sa.Divide(numerator, denominator)
    # save intermidiate before reclass
    NDWI_eq.save(os.path.join(folder_path, 'raw_ndwi.TIF'))

    # Reclassify to get a mask
    result = Reclassify(NDWI_eq, "Value", RemapRange([[-2, 0, 0], [0.01, 2, 1]]))

    #Saving output to result output you specified above
    try:
        result.save(output_name)
        log.debug("NDWI Successful: {}".format(output_name))
        result = output_name
    except:
        log.debug("NDWI Unsuccessful: {}".format(result))
        result = 0

    return result

def get_raster_count(args):
    """
    Subfunction to deal with ensuring there's a RAT and then getting the cell
    count out of the table using a cursor.
    Accepts raster.
    Returns integer value of total cell count or -1.
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    result = -1
```

```
    x = args
    # see if it's all null before the bother
    null_result = arcpy.GetRasterProperties_management(x, 'ALLNODATA')
    # make result useful
    null = bool(int(null_result.getOutput(0)))
    if null:
        result = 0
    else:
    # bother
        if x.hasRAT:
            logger.debug('{} has RAT'.format(x))
        else:
            try:
                arcpy.BuildRasterAttributeTable_management(x, "OVERWRITE")
            except:
                logger.debug('{} failed to build RAT'.format(x))

        # now RAT is built
        total_count = 0
        with arcpy.da.SearchCursor(os.path.join(x.path, x.name), ["VALUE", "COUNT"]) as rows:
            for row in rows:
                print row
                # nodata isn't in the RAT, so no need to filter it out
                # but we do skip the 0's
                if row[0] > 0:
                    total_count += row[1]
        result = float(total_count)

    log.debug('{} result = {}'.format(x.name, result))
    return result

def not_cloud_mask(*args):
    """
    0 = cloud, 1 = not cloud, 2 = ambig
    Return path of mask?
    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug('with {}'.format(args))
    result = -1

    folder_path = args[0]
    images = [os.path.join(folder_path, i) for i in args[1]]
    #print images
    B2 = Raster(images[0])
    B3 = Raster(images[1])
    B4 = Raster(images[2])
    B5 = Raster(images[3])
```

```
B6 = Raster(images[4])

# save final mask in folder_path dir
# all the other stuff can be in the cloud_mask dir
working_path = os.path.join(folder_path, settings['Cloud Dir'])
# make sure the output dir exists, if not, make
out = working_path
if not os.path.exists(out):
   os.makedirs(out)

#total_count = 0
nodata_value = arcpy.Describe(B2).noDataValue
output_name = os.path.join(folder_path, 'not_cloud.TIF')
agg_name = os.path.join(working_path, 'agg_mask.TIF')
name_0 = os.path.join(working_path, 'cloud.TIF')
name_1 = os.path.join(working_path, 'not_cloud.TIF')
name_2 = os.path.join(working_path, 'ambig.TIF')

sql_exp = """{} > {}""".format(arcpy.AddFieldDelimiters(B2, 'VALUE'), nodata_value)
log.debug('test sql statement: {}'.format(sql_exp))

# test_raster just to get in count of data pixels
test_raster = Test(B2, sql_exp)
test_raster.save(os.path.join(working_path, 'test.TIF'))
total_count = get_raster_count(test_raster)

# set test_image to 0 to seed masks
test_raster = Con(test_raster > 0, 0)

# number of raster cells
snow_count = 0
hot_count = 0
cold_count = 0
desert_count = 0

# filter 1: B3 < .08 = 1/not cloud
f1 = Con(B3 < .08, 1, 0)
f1.save(os.path.join(working_path, 'f1.TIF'))
# add f1 to an aggregate mask
# add to aggregate (binary mask, assigned a category or not)
agg = BooleanOr(f1, test_raster)
agg.save(os.path.join(working_path, 'agg1.TIF'))

# running sums
sum_0 = 0
sum_1 = get_raster_count(agg)
sum_2 = 0
log.debug('f1- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
```

```
   sum_1 / total_count, sum_2 / total_count))

# filter 2: NSDI (B2 - B5)/(B2 + B5). >.7 = 1/not cloud (get count of snow?)
NSDI = ((B2 - B5)/(B2 + B5))
sf2 = Con(NSDI > 0.7, 1, 0)
sf2.save(os.path.join(working_path, 'sf2.TIF'))

# just f2 additions
f2 = Diff(sf2, agg)
f2.save(os.path.join(working_path, 'f2.TIF'))

# get snow count
get_raster_count(f2)

# use aggregate + selection
agg = BooleanOr(agg, sf2)
agg.save(os.path.join(working_path, 'agg2.TIF'))
# running sums
sum_0 = 0
sum_1 = get_raster_count(agg)
sum_2 = 0
log.debug('f2- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

# filter 3: B6 > 300K = 1
sf3 = Con(B6 > 300, 1, 0)
sf3.save(os.path.join(working_path, 'sf3.TIF'))
f3 = Diff(sf3, agg)
f3.save(os.path.join(working_path, 'f3.TIF'))
agg = BooleanOr(agg, sf3)
agg.save(os.path.join(working_path, 'agg3.TIF'))

# copy not-cloud aggregate to (name_1) since switching to detect Ambig
mask_1 = agg
# running sums
sum_0 = 0
sum_1 = get_raster_count(mask_1)
sum_2 = 0
log.debug('f3- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

# filter 4: B5/6 Composite (1 - B5) * B6 > 225 = 2
f4 = ((1 - B5) * B6)
f4.save(os.path.join(working_path, 'rf4.TIF'))
sf4 = Con(((1 - B5) * B6) > 225, 1, 0)
sf4.save(os.path.join(working_path, 'sf4.TIF'))

f4 = Diff(sf4, agg)
```

```
    f4.save(os.path.join(working_path, 'f4.TIF'))

    # add to aggregate
    agg = BooleanOr(agg, sf4)
    agg.save(os.path.join(working_path, 'agg4.TIF'))
    # start to cloud name_1 (ambig)
    mask_2 = f4
    mask_2.save(os.path.join(working_path, 'f4_a.TIF'))
    # running sums
    sum_0 = 0
    sum_1 = get_raster_count(mask_1)
    sum_2 = get_raster_count(mask_2)
    print sum_1, sum_2, total_count, sum_2/total_count
    log.debug('f4- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
       sum_1 / total_count, sum_2 / total_count))

    # filter 5: B3/4 Ratio (B4/B3) > 2.0 = 2
    sf5 = Con((B4/B3) > 2.0, 1, 0)
    sf5.save(os.path.join(working_path, 'sf5.TIF'))
    f5 = Diff(sf5, agg)
    f5.save(os.path.join(working_path, 'f5.TIF'))
    # add to aggregate (binary mask, assigned a category or not)
    agg = BooleanOr(agg, f5)
    agg.save(os.path.join(working_path, 'agg5.TIF'))
    # add to cloud name_1 (ambig)
    mask_2 = BooleanOr(mask_2, f5)
    mask_2.save(os.path.join(working_path, 'f5_a.TIF'))
    # running sums
    print 'mask checks:', type(mask_1), mask_1.path, mask_1.name, type(mask_2), mask_2.path,
mask_2.name
    sum_0 = 0
    sum_1 = get_raster_count(mask_1)
    sum_2 = get_raster_count(mask_2)
    log.debug('f5- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
       sum_1 / total_count, sum_2 / total_count))

    # filter 6: B4/2 Ratio (B4/B2) > 2.0 = 2
    sf6 = Con((B4/B2) > 2.0, 1, 0)
    sf6.save(os.path.join(working_path, 'sf6.TIF'))
    f6 = Diff(sf6, agg)
    f6.save(os.path.join(working_path, 'f6.TIF'))

    # add to aggregate
    agg = BooleanOr(agg, f6)
    # add to cloud name_1
    mask_2 = BooleanOr(f6, mask_2)
    mask_2.save(os.path.join(working_path, 'f6_a.TIF'))
    # running sums
```

```
sum_0 = 0
sum_1 = get_raster_count(mask_1)
sum_2 = get_raster_count(mask_2)
log.debug('f6- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

# filter 7: B4/5 Ratio (B4/B5) > 1.0 = 2 (get count of desert pixels)
sf7 = Con((B4/B5) > 1.0, 1, 0)
sf7.save(os.path.join(working_path, 'sf7.TIF'))
f7 = Diff(sf6, agg)
f7.save(os.path.join(working_path, 'f7.TIF'))
desert_count = get_raster_count(f7)
log.debug('desert_count: {}'.format(desert_count))

# add to aggregate (binary mask, assigned a category or not)
agg = BooleanOr(f7, agg)
# add to cloud name_1
mask_2 = BooleanOr(f7, mask_2)
mask_2.save(os.path.join(working_path, 'f7_a.TIF'))
# running sums
sum_0 = 0
sum_1 = get_raster_count(mask_1)
sum_2 = get_raster_count(mask_2)
log.debug('f7- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

# filter 8: Everything remaining = 0 (hot and cold clouds)
sf8 = BooleanNot(agg)
sf8.save(os.path.join(working_path, 'sf8.TIF'))
f8 = Diff(sf8, agg)
f8.save(os.path.join(working_path, 'f8.TIF'))

mask_0 = sf8
# running sums
sum_0 = get_raster_count(mask_0)
sum_1 = get_raster_count(mask_1)
sum_2 = get_raster_count(mask_2)
log.debug('f8- {:.3f} cloud. {:.3f} not cloud. {:.3f} ambig.'.format(sum_0 / total_count,  \
   sum_1 / total_count, sum_2 / total_count))

#finalize masks
log.debug('saving masks')
agg.save(agg_name)
mask_0.save(name_0)
mask_1.save(name_1)
mask_2.save(name_2)

# pass 2: only fires if there's clouds and all the 3 checks fail
```

```
    # if mask_1 is all 0: cloudfree, else try checks
    c_mask_0 = SetNull(mask_0, 1, "VALUE = 0")
    null_result = arcpy.GetRasterProperties_management(mask_0, 'ALLNODATA')
    null = bool(int(null_result.getOutput(0)))
    log.debug('{} all no data check: {}'.format(f8.name, null))
    if null:
    # no clouds = no pass 2
        log.debug('no clouds, no pass 2')
        result = 1
    else:
        check = 0   # if check == 3, do pass 2

        # if desert_index > 0.5: increment check
        if desert_count == 0:
            desert_index = 0
        else:
            #print desert_count, type(desert_count), total_count, type(total_count)
            desert_index = desert_count / total_count
        if desert_count > 0.5:
            check += 1
            log.debug('di did trip'.format(desert_index))
        else:
            log.debug('di did not trip'.format(desert_index))

        # if cold clouds are < .4%, skip pass 2
        if cold_count == 0:
            cold_index = 0
        else:
            # use B6 since it's the truest count of final image/filter
            cold_index = cold_count / get_raster_count(B6)
        if cold_index  < .004:
            log.debug('ci count did not trip'.format(cold_index))
        else:
            check += 1
            log.debug('ci did trip'.format(cold_index))

        # if mean temp of cold class < 295K
        #cold_mean = GetRasterProperties_management (in_raster, {property_type}, {band_index}) /
cold_count
        cold_mean_result = arcpy.GetRasterProperties_management(ExtractByMask(B6, f8), 'MEAN')
    # maybe extract B6 using Mask_0, and then get average from this.
        cold_mean = float(cold_mean_result.getOutput(0))
    # but just to test, really have to decide which.
        if cold_mean < 295:
            check += 1
            log.debug('cm count did trip'.format(cold_mean))
        else:
            log.debug('cm did not trip'.format(cold_mean))
```

```
if check == 3:
    log.debug('need pass 2')
    # setup some vars and things

    # split cloud into warm and cold for p2
    # B5/6 Composite (1 - B5) * B6 > 210 = warm, rest are cold
    warm_cloud = Con((f8 * ((1 - B5) * B6)) > 210, 1, 0)
    warm_cloud.save(os.path.join(working_path, 'warm_cloud.TIF'))
    warm_count = get_raster_count(hot_cloud)
    cold_cloud = Con((f8 * ((1 - B5) * B6)) <= 210, 1, 0)
    cold_cloud.save(os.path.join(working_path, 'cold_cloud.TIF'))
    cold_count = get_raster_count(cold_cloud)

    #selection of which cloud groups to process
    if snow_count:
        snow_pct = snow_count / total_count
        log.debug('snow_pct: {}'.format(snow_pct))
        # just assign the selection to a var
        if snow_pct < .01:
            # snow-free, use hot_cloud and cold_cloud AKA mask_0
            log.debug('snow free, use mask_0')
            test_clouds = mask_0
        else:
            # use cold cloud only only
            log.debug('snow found, use cold and move warm to ambig')
            test_clouds = cold_cloud
            # reassign warm_cloud as ambig/mask_2
            mask_2 = BooleanOr(mask_2, warm_cloud)
            mask_2.save(os.path.join(working_path, 'ambigP2.TIF'))
    else:
        log.warning('no snow_count: {}'.format(snow_count))

    # with selection, run through threshold evaluation
    raster_stats = CalculateStatistics_management(test_clouds, "MINIMUM")
    print 'cloud min', raster_stats.getOutput(0)

    test_arr = np.array(test_clouds)
    min = np.amax(test_arr)
    max = np.amin(test_arr)
    std_dev = np.std(test_arr)
    mean = np.mean(test_arr)
    n = 0
    for i in test_arr:
        n += (i - mean)^3
    skew = n / std_dev
    skew = sp.stats.skew(test_arr)
    log.debug('min {}. max {}. mean {}. std dev {}. skew {}'.format( \
```

```
   min, max, std_dev, skew))

max_threshold = (max - min) * .9875 + min
high_threshold = (max - min) * .975 + min
low_threshold = (max - min) * .825+ min

if skew > 0:
   # no adjustment to thresholds
   log.debug('no shift')
else:
   # adjust thresholds up
   shift_factor = skew * std_dev
   log.debug('shift factor: {}'.format(shift_factor))
   low_threshold = low_threshold * shift_factor
   high_threshold = high_threshold * shift_factor

   # back high down if too high
   if high_threshold > max_threshold:
      high_threshold = max_threshold
      log.debug('high_threshold reassigned: {}'.format(high_threshold))
   else:
      log.debug('high_threshold passed: {}'.format(high_threshold))

# Evaluate "termal effects"
g1 = Con(test_clouds < high_threshold, 1, 0)
g1 = Con(test_clouds < low_threshold, 0, g1)
g2 = Con(g1 < low_threshold, 1, 0)

# compute stats for upper
g1_count = get_raster_count(g1)
g12 = np.count_nonzero(~np.isnan(np.array(g1)))
print 'count test: {} vs {}'.format(g1_count, g12)
g1_pct = g1_count / total_count
g1_mean = np.mean(g1)

if g1_pct > .40 or g1_mean > 295:
   # then g1 are classified non-cloud
   mask_1 = BooleanOr(mask_1, g1)
   log.debug('rejecting g1')
   # continue to evaluate g2
   # compute stats for lower
   g2_count = get_raster_count(g2)
   g2_pct = g2_count / total_count
   g2_mean = np.mean(g2)

   if g2_pct > .40 or g2_mean > 295:
      # then all ambig are scrapped
      mask_1 = BooleanOr(mask_1, mask_2)
```

```
                log.debug('rejecting all ambig')
            else:
                # accept the lower group into cloud
                mask_0 = BooleanOr(mask_0, g2)
                log.debug('uniting the lower with cloud')
        else:
            # unite all 3 cloud classes
            log.debug('uniting the 3 cloud classes')
            mask_0 = BooleanOr(mask_0, mask_2)

        # CHEQUES
        mask_0.save(os.path.join(working_path, 'cloudP2.TIF'))
    else:
        log.debug('at least one check was breeched, no Pass 2.')
        mask_0 = BooleanOr(test_raster, B6)
    # save final masks
    mask_0.save(os.path.join(working_path, 'cloudP2.TIF'))
    mask_1.save(os.path.join(working_path, 'not_cloudP2.TIF'))
    result = 2
  # and then cloud holes filled in?
    mask_1.save(output_name)
  log.debug('not cloud mask named {}'.format(output_name))
  return output_name

def select_values(*args):
  """
  Get values from points around sample sites, if masks say its ok.
  Return as a table in Proccessed dir?
  """
  log = logger.getChild(inspect.currentframe().f_code.co_name)
  log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
  folder_path = args[0]
  #need joined to path corrected_scenes = args[1]
  corrected_scenes = [os.path.join(folder_path, i) for i in sorted(args[1])]
  # now that htey are pathed, append not_cloud mask in
  corrected_scenes.append(args[2])
  log.debug('with {} ~ {}'.format(folder_path, corrected_scenes))
  result = []

  extract_dir = os.path.join(working_dir, settings['Extr Dir'], os.path.basename(folder_path))
  log.debug('extract dir {}'.format(extract_dir))

  # Get the sample point shapefile out of the dir corresponding with first 2 of this folder
  clip_dir = os.path.basename(folder_path)[:2]
  point_file = os.path.join(working_dir, settings['Clip Dir'], (clip_dir +'_pnts.shp'))
  log.debug('selecting sample points from dir: {}'.format(point_file))
  # have SR ready for features made
  spatial_reference = arcpy.Describe(point_file).spatialReference
```

```
fields = ['SITE_ID', 'X', 'Y']
# make sure there's things
count = int(arcpy.GetCount_management(point_file).getOutput(0))
log.debug('len of feature {}: {}'.format(point_file, count))
if count:
   # list of files to aggregate after cursor
   extract_list = []

   # make output dir
   out = extract_dir
   if not os.path.exists(out):
      os.makedirs(out)

   # use sample on the XX_pnt file to get cell address
   xy_table = (os.path.join(extract_dir, 'xy.dbf'))
   Sample(corrected_scenes[0], point_file, xy_table)
   # join the SITE_ID from original file to xy_table
   arcpy.JoinField_management (xy_table, clip_dir +'_pnts', point_file, 'FID', 'SITE_ID')

   # iterate by site_id
   with arcpy.da.SearchCursor(xy_table, fields) as cursor:
      for row in cursor:
         print row
         log.debug('starting {}, {}/{}'.format(row[0], row[1], row[2]))
         # validate shortens, we just need dash out of name
         clean_siteid = row[0].replace('-', '_').strip()
         # get points, make into window coords
         points = []
         points.append([row[1], row[2]])
         window_coords = utm_window(points)
         log.debug('window coords: {}'.format(window_coords))

         # final extract table name
         output_name = os.path.join(extract_dir, clean_siteid.partition('_')[2] +'.dbf')
         extract_list.append(output_name)
         name = os.path.join(extract_dir, clean_siteid)
         log.debug('extract name: {}, pointfile name: {}'.format(output_name, name))
         # pointfile of window points
         pf_name = make_pointfile(window_coords, name)
         # apply SR from source file
         arcpy.DefineProjection_management(pf_name, spatial_reference)
         # set the id field to the site_id

         # Set local variables
         inFeatures = pf_name
         fieldName1 = "SID"
         fieldType = 'TEXT'
```

```
            fieldLength = 20

            # Sample
            Sample(corrected_scenes, pf_name, output_name)
            logger.debug('sampled {}'.format(output_name))
            # read sampled values
            arcpy.AddField_management(output_name, fieldName1, fieldType, \
               '#', '#', fieldLength)
            logger.debug('just added field {}, goign to populate it with {}' \
               .format(fieldName1, clean_siteid))
            with arcpy.da.UpdateCursor(output_name, '*') as cur:
               for row in cur:
                  # should be last one since most recently appended
                  row[-1] = clean_siteid
                  #print row
                  cur.updateRow(row)

      # aggregated sample file
      agg_ext = os.path.join(extract_dir, 'agg.dbf')
      # merge data into single file
      arcpy.Merge_management(extract_list, agg_ext)
      logger.debug('merged all of {}'.format(clean_siteid.partition('_')[0]))

   else:
      log.debug('{} was empty'.format(point_file))

   return result

def aggregator(arg):
   """
   Accepts output dir and aggrated file name.
   Reads them all and puts them into a single CSV in the Extracts dir
   """
   log = logger.getChild(inspect.currentframe().f_code.co_name)
   log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
   log.debug('with {}'.format(arg))
   result = []

   extract_dir = os.path.join(os.getcwd(), settings['Extr Dir'])

   # get folders in the Extract dir
   folders_in_dir = [i for i in listdir(extract_dir) if isdir(os.path.join(extract_dir, i))]
   log.debug('folders_in_dir len & (short) list: {} {}'.format(len(folders_in_dir), folders_in_dir[:3]))

   # initialize output
   output = []

   # walk dirs and open the file
```

```
    for f in folders_in_dir:
        logger.debug('iterating with {}'.format(f))
       # if settings['Agg File'] in listdir(os.path.join(extract_dir, f)):
        f_path = os.path.join(extract_dir, f, settings['Agg File'])

    # need to pick out some field headings here, and just read those in cursor
        # if first folder, initialize field names
        if output == []:
            field_names = [i.name for i in arcpy.ListFields(f_path)]
            field_names.insert(0, 'folder name')
            log.debug('field names {}'.format( field_names))
            output.append(field_names)
    # first read, append header selection
        try:
            with arcpy.da.SearchCursor(f_path, "*") as cursor:
                for row in cursor:
                    x = [i for i in row]
                    x.insert(0, f)
                    output.append(x)
        except:
            log.debug('{} did not have a {}'.format(f, settings['Agg File']))

    # write output into source folder
    with open(os.path.join(extract_dir, settings['Agg Output']), 'wb') as file:
        writer = csv.writer(file)
        writer.writerows(output)
        logging.info('output saved as {}\{}'.format(extract_dir, settings['Agg Output']))

    return result


#//////////////////////////   Correction Funcion  ///////////////////////////#

def main():
    pass

# Setup Logger
logger = logging.getLogger(os.path.basename(__file__))
logger.setLevel(logging.DEBUG)

formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
format2 = logging.Formatter('%(levelname)s - %(message)s')

# Logging to file
fh = logging.FileHandler(log_location)
fh.setLevel(logging.DEBUG)
fh.setFormatter(formatter)
logger.addHandler(fh)
```

```
# Logging to conlsole/standard output
ch = logging.StreamHandler()
ch.setLevel(output_level)
ch.setFormatter(formatter)
logger.addHandler(ch)

# open the metadata dict
with open(os.path.join(settings['Output Dir'], settings['MD file']), 'rb') as handle:
   Correction_Parameters = pickle.loads(handle.read())

# open the solar distance file, read into a dict
with open(os.path.join(working_dir, settings['SD File']), 'rb') as handle:
   reader = csv.reader(handle)
   for row in islice(reader, 1, None): #skip header row
      sun_d[row[0]] = float(row[1])

# read the folders to process in the Processed folder
image_dir = os.path.join(os.getcwd(), settings['Output Dir'])
logger.debug('processed_dir is: {}'.format(image_dir))

folders_in_dir = [i for i in listdir(image_dir) if isdir(os.path.join(image_dir, i))]
logger.debug('folders_in_dir len & list: {} {}'.format(len(folders_in_dir), folders_in_dir))

if __name__ == '__main__':
   main()

# cycle thorugh contents of Process folder
for folder in folders_in_dir:
   folder_path = os.path.join(working_dir, settings['Output Dir'], folder)
   logger.debug('folder path: {}'.format(folder_path))
   # get list of scenes with clip prefix and tif ext
   scenes_in_dir = [i for i in listdir(folder_path) if i[:2] == settings['Clip Prefix'] \
      and i.partition('.')[2] == 'TIF']
   logger.debug('scenes_in_dir {} len & list: {} {}'.format(folder, len(scenes_in_dir), scenes_in_dir))

   scene_key = set([i.split('_')[1] for i in scenes_in_dir])
   logger.debug('scene_key {}'.format(scene_key))

   # load a subset of Corr Para for this scene
   scene_d = dict((i, Correction_Parameters[i]) for i in Correction_Parameters if i in scene_key)
   # having to use scene name sucks, since it's arelady pulled just "what we want"
   scene_d = scene_d.values()[0]
   logger.debug('scene parameters {}'.format(scene_d))

# code modified from ...
   for scene in scenes_in_dir:
      scene_path = os.path.join(folder_path, scene)
```

```
    #scene_path = os.path.join(working_dir, settings['Output Dir'], folder, scene)
    band = scene.split('_')[2].split('.')[0]
    logger.debug('reading {}, band = {}'.format(scene_path, band))
    # check if the image contains any data
    radiance_ras = Raster(scene_path)
    # see if the image is all NoData
    #   this returns a result object... part of which is string of result that needs converted
    #   to binary to be useful
    nd_check = arcpy.GetRasterProperties_management (radiance_ras, 'ALLNODATA')
    logger.debug('nd_check = {}'.format(nd_check))
    t = bool(int(nd_check.getOutput(0)))
    print 'int of t', t, type(t), bool(t)

    if t:
       logger.warning('image {} is all NoData, no processing'.format(scene))
    else:
       #radianceRaster = calcRadiance(LMAX, LMIN, QCALMAX, QCALMIN, BANDFILE, outfolder)
       # (LMAX, LMIN, QCALMAX, QCALMIN, QCAL, outfolder)
       radiance_ras = calc_radiance(scene_d['lmax'][band], scene_d['lmin'][band], \
          scene_d['qc_lmax'][band], scene_d['qc_lmin'][band], scene_path , folder_path)

       # B6 > Temp, everything else goes for Correction
       if band == 'B6':
          test = b6_to_temp(scene_d['sat_id'], radiance_ras)
          if test:
             logger.debug('{} successfully converted to kelvins'.format(scene_path))
          else:
             logger.debug('{} failed to convert to kelvins'.format(scene_path))
       else:
          # DNMin is in DN. Has to be converted to Radiance seperatley form scene.
          # COST DOS correction (includes scaling DNMin)
          dnmin_as_radiance = 0.0
          if Do_COSTDOS:
             offset = (scene_d['lmax'][band] - scene_d['lmin'][band])/(scene_d['qc_lmax'][band] -
scene_d['qc_lmin'][band])
             HLmin = (scene_d['qc_lmin'][band] + scene_d['dnmin'][band] * offset)
             logger.debug('HLmin: {}'.format(HLmin))

             # now correct it
             solarZenith = (90.0 - float(solarElevation))* (math.pi / 180)    #Converted from degrees to
radians
             logger.debug('solar zenith in rad: {}'.format(solarZenith))
             logger.debug('n = {}'.format((0.01 + get_ESUN(band, scene_d['sat_id']) *
math.cos(solarZenith))))
             logger.debug('d = {}'.format((math.pow(sun_d[scene_d['julian_date'].lstrip('0')], 2) *
math.pi)))
             L1percent = ((0.01 + get_ESUN(band, scene_d['sat_id']) * math.cos(solarZenith)) / \
                (math.pow(sun_d[scene_d['julian_date'].lstrip('0')], 2) * math.pi))
```

```
            logger.debug('L1percent: {}'.format(L1percent))
            dnmin_as_radiance = HLmin - L1percent
            logger.debug('DNMin of {} converted to radiance {}'.format(scene_d['dnmin'][band],
dnmin_as_radiance))
         else:
            logger.debug('DNMin is 0 (preset)')
         reflectanceRaster = calc_reflectance(sun_d[scene_d['julian_date'].lstrip('0')], \
            get_ESUN(band, scene_d['sat_id']), scene_d['sun_elevation'], radiance_ras,
dnmin_as_radiance, folder_path)

   # now from the corrected scenes grab band 2 & 5 (if any)
   corrected_prefix = settings['Corr Prefix'] + settings['Rad Prefix'] + settings['Clip Prefix']
   corrected_scenes = [i for i in listdir(folder_path) if corrected_prefix in i \
      and i.partition('.')[2] == 'TIF']
   logger.debug('corrected scene list 2: {}'.format(corrected_scenes))
   # need to get TRC in there for cloud_mask

   if len(corrected_scenes) > 2:
      B2 = [i for i in corrected_scenes if settings['Green Band'] in i][0]
      B5 = [i for i in corrected_scenes if settings['SWIR Band'] in i][0]

      # send to ndwi
      if B2 and B5:
         ndwi = ndwi_mask(folder_path, B2, B5)
      else:
         logger.warning('ndwi failed to process: {} {}'.format(B2, B5))
      B3 = [i for i in corrected_scenes if settings['Red Band'] in i][0]
      B4 = [i for i in corrected_scenes if settings['NIR Band'] in i][0]
      unwanted_thermal = 'VCID_2'
      thermal_prefix = settings['Temp Prefix'] + settings['Rad Prefix'] + settings['Clip Prefix']
      B6 = [i for i in listdir(folder_path) if thermal_prefix in i \
         and i.partition('.')[2] == 'TIF' and unwanted_thermal not in i][0]
      logger.debug('found thermal band: {}'.format(B6))
      corrected_scenes.append(B6)

      # do cloud mask
      cloud = not_cloud_mask(folder_path, [B2, B3, B4, B5, B6])

      # ndwi gets caught in corrected scenes!
      select_values(folder_path, corrected_scenes, cloud)
      # which searches out the masks from before to exlude from extraction
   else:
      logger.debug('No corrected scenes to process: {}'.format(corrected_scenes))
aggregator(1)

logger.debug(u'finished')
```

BUMP and Processed Data Assembly

```
import os
from os import listdir
from os.path import isfile, join, isdir
import csv
from datetime import date, datetime, timedelta
import logging
import inspect

settings = {
'Image List': 'Master List.csv',
'Processed Dir': 'Output',
'Processed List': 'Aggregated.csv',
'Bump Data': 'Bump Data.csv',
'Exclude Site': 'B',
'date allowance': timedelta(days=1),
'Final Output': 'final.csv'
}

working_dir = os.getcwd()

def read_csv(args):
    """

    """
    log = logger.getChild(inspect.currentframe().f_code.co_name)
    log.info(u'Initializing {}'.format(inspect.currentframe().f_code.co_name))
    log.debug(u'with {}'.format(args))
    result = []

    with open(args, 'rb') as csvfile:
        log.debug(u'Reading {} as csvfile.'.format(args))
        reader = csv.reader(csvfile)
        for row in reader:
            result.append(row)
    return result


def main():
    pass

if __name__ == '__main__':
    main()

# Setup Logger
logger = logging.getLogger(os.path.basename(__file__))
logger.setLevel(logging.DEBUG)
```

```
formatter = logging.Formatter('%(asctime)s - %(name)s - %(levelname)s - %(message)s')
format2 = logging.Formatter('%(levelname)s - %(message)s')

# Logging to conlsole/standard output
ch = logging.StreamHandler()
ch.setLevel(logging.DEBUG)
ch.setFormatter(formatter)
logger.addHandler(ch)

# Read Master list
master_list = read_csv(os.path.join(working_dir, settings['Image List']))

# Read Processed image data
aggregated_list = read_csv(os.path.join(working_dir, settings['Processed Dir'], settings['Processed List']))
result = []
# initialize final output headers
merge_result = [['folder name', 'lake name', 'scene', 'scene date', 'site id', 'B1', 'B2', 'B3', \
   'B4', 'B5', 'B7', 'other result','Corrected Chlorophyl A', 'Turbidity, Field', 'season']]
# initialize mask check values, 0 = do not use
test_value = ['0.0', '0.0']

# Use folder name to select data for each scene out of processed data
folder_list = [i[0] for i in master_list[1:]]
for f in folder_list:
   logger.debug('iterating folder {}'.format(f))
   data_selection = [i for i in aggregated_list if f in i]
   logger.debug('folder data selection: {}'.format(data_selection))
   if data_selection:
      # iterate by scene and evaluate final values (- Bottom readings)
      site_list = set([i[-1] for i in data_selection if settings['Exclude Site'] not in i[-1]])
      logger.debug('site_list: {}'.format(site_list))
      line_seed =[i for i in master_list[1:] if f in i][0]
      for s in site_list:
         logger.debug('processing site {}'.format(s))
         # seed is Folder Name, Lake Name, Scene Name, and Scene Date
         line = []
         line.extend(line_seed)
         logger.debug('line initialized: {}'.format(line))
         # append site_id converting '_' back to '-'
         line.append(s.replace('_', '-'))
         site_selection = [i for i in data_selection if s in i]
         logger.debug('site_selection: {}/{}'.format(len(site_selection), site_selection))
         test_masks = [i[11:13] for i in s]
         if test_value in test_masks:
            logger.debug('all masks not ok')
            line.append('X')
            line.append('X')
            line.append('X')
```

```
            line.append('X')
            line.append('X')
            line.append('X')
            line.append('reason code')
        else:
            # take a mean of the 4 desired bands
            index = [5, 6, 7, 8, 9, 10]
            for j in index:
                num = [i[j] for i in site_selection]
                # magically turn strings into numbers for division
                L = [float(n) for n in num if n]
                avg = sum(L)/len(L) if L else '-'
                logger.debug('avg: {}'.format(avg))
                line.append(avg)
            line.append('np')
        # Check if we should ship writing due to all bands being NoData
        if all(line[5:11]):
            merge_result.append(line)
        else:
            logger.debug('nd for all bands, not writing to output')
        logger.debug('line: {}'.format(line))
    else:
        logger.debug('{} had no output'.format(f))
logger.debug('master list and aggregated merged result: {}'.format(merge_result))

# open bump metadata
bump_list = read_csv(os.path.join(working_dir, settings['Processed Dir'], settings['Bump Data']))
# for each line in result compare to site & date in bump list, appending sample data
for (x, i) in enumerate(merge_result):
    # skip header
    if x > 0:
        r = i
        logger.debug('seeded r {}'.format(r))
        site = i[4]
        date = i[3]
        # dates are in '02/25/2001' format
        do = datetime.strptime(date, "%m/%d/%Y").date()
        date_check = []
        date_check.append(do)
        date_check.extend([do - settings['date allowance']])
        date_check.extend([do + settings['date allowance']])
        logger.debug('date objects for check {}'.format(date_check))
        # bump list has date as string, so put these back to string
        acceptable_dates = [datetime.strftime(d, "%m/%d/%Y") for d in date_check]
        logger.debug('acceptable string dates: {}'.format(acceptable_dates))
        logger.debug('check for {} {} in bump'.format(site, date))
        selection = [i for i in bump_list if site in i and i[3] in acceptable_dates]
        logger.debug('selection of bump: {}'.format(selection)) # try for id + date range
```

```
        if selection:
            # each data seems to have its own row... assume there are no dupes here
            chlor = ''
            turb = ''
            for j in selection:
                logger.debug('checking {}'.format(j[5:7]))
                if j[5]:
                    chlor = j[5]
                if j[6]:
                    turb = j[6]
            r.extend([chlor, turb])

            logger.debug('month: {}'.format(do.month))
            if do.month > 9:
                season = 'fall'
            elif do.month > 6:
                season = 'summer'
            elif do.month > 3:
                season = 'spring'
            elif do.month > 0:
                season = 'winter'
            else:
                season = 'error'
            r.append(season)
            logger.debug('keeping line {}'.format(r))
            result.append(r)
        else:
            logger.warning('delete due to failing to match processed scene to a bump sample date')
    else:
        logger.debug('seed result with header: {}'.format(i))
        result.append(i)

# write output into output folder
output_path = working_dir, settings['Processed Dir'], settings['Final Output']
logger.debug('final output: {}'.format(output_path))
type(output_path)
#with open(os.path.join(output_path), 'wb') as file:
with open(os.path.join(working_dir, settings['Processed Dir'], settings['Final Output']), 'wb') as file:
    writer = csv.writer(file)
    writer.writerows(result)
    logger.info(u'result saved as {}'.format(output_path))

logger.debug('result {}'.format(result))

logger.info(u'finished')
```