I would suggest that things like the fields being read from each file be configurable. So the flow of the skeleton code would be:

* Run the CONTROL\_MODULE()
* This will first call READ\_CONFIG(), which reads in a plain text configuration file. Inside that config file are things like the start and end dates for the period we want to analyze (start small!), latitude and longitude limits for the analysis, a set of flags that say which diagnostic plots should be created (i.e. should we create a plot of the lightning locations every time we read a new LIS file? Make a flag for that). This config file can be extended. I suggest for each of the different datasets being included that the list of variables we want from each dataset be written into the config file. That way, you can specify what data you want each time you run the control module.
* Once we’ve read config, start stepping through the dates one by one and reading the LIS files first, READ\_LIS(). This will give a set of latitude/longitude points where lightning flashes occurred. If you read an LIS file successfully, call SELECT\_DATA(). In that function, we can query the TROPOMI, MODIS, and VIIRS datasets in the vicinity (time and location) of each flash and read the data as appropriate. Again, each of these datasets will receive a list of variables from config that instructs the READ\_\* functions what to store.
* Still within the time step loop, we next want to call CENTRE\_FLASH(). This function is the regridding (aka interpolation) that will take the four sets of variables that have now been read in and line them up on a common grid with the flash location and time at the origin. It’s possible that we could specify the resolution of the common grid within the config file.
* Continue time stepping through each LIS file and repeat the above steps until we reach the end date. At this point, call WRITE\_OUTPUT() to create an output file that will store the new data product. Remember to pass any relevant dimensions and metadata for traceability. This is another place where PLOT\_DIAGNOSTIC() could be called.

This is just my interpretation of how the code might run; if you have ideas on how to speed it up (looping can be slow) or to incorporate more flexibility into the design, then have at it. We can present the functions with their documentation (docstrings) tonight and see what our coders can come up with.

def read\_lis(filename, fieldlist):

    """

    Reads an LIS file given by its FILENAME. LIS files are .nc files. Attempts to read each of the fields in FIELDLIST and return the data they contain.

    FILENAME is the full name and path of the netCDF file to be read. This function assumes the file is of the same structure as the ISS LIS data files

    obtained from (insert full name of the dataset here). If the file is unable to be read, return -1.

    FIELDLIST is a list of the data fields to be read, including their full path in the hierarchical file. The fields in LIS files are all at the top level so the

    path is straightforward, and the fieldlist will just contain the variable names. If a field requested in the list is unable to be read, return -1 as the

    value of that field.

    Returns a dictionary with keys corresponding to the names of the data fields given in FIELDLIST, and values for each of these keys that are numpy

    arrays containing the data in the respective field.

    """

    # the code

def read\_tropomi():

    """

    Reads a TROPOMI file given by its filename. TROPOMI files are .nc files.

    """

    # the code

def read\_modis():

    """

    Reads a MODIS file given by its filename. MODIS files are .h5 files.

    """

    # the code

def read\_viirs():

    """

    Reads a VIIRS file given by its filename. VIIRS files are .h5 files.

    """

    # the code

def select\_data():

    """docstr"""

    # the code

def centre\_flash():

    """docstr"""

    # the code

def write\_output():

    """docstr"""

    # the code

def plot\_diagnostic():

    """docstr"""

    # the code

def control\_module():

    """

    This function runs the project.

    """

    # the code

def read\_config():

    """

    Reads the config file for the project.

    """

    # the code