```python
import numpy as np
import matplotlib.pyplot as plt
import astropy
from astropy.io import fits
import matplotlib.cm as cm
import scipy.signal
import glob
import os
from astropy.stats import sigma_clip
import time
from PIL import Image

def mediancombine(filelist, newname):
    # assign the number of files considered to a variable n
    n = len(filelist)
    biasheader = fits.getheader(filelist[0])

    # pull out the first fits file in the list as an array
    first_frame_data = fits.getdata(filelist[0])

    # pull out the dimensions of each fits file, assuming they are all identical
    imsize_y, imsize_x = first_frame_data.shape

    # create an empty 3D array with axes 0, 1 the dimensions of the images and axis 2 the number of files considered
    fits_stack = np.zeros((imsize_y, imsize_x , n))

    # loop over each file and fill the array with the image data, for each index in axis 2
    for ii in range(0, n):
        im = fits.getdata(filelist[ii])
        fits_stack[:,:,ii] = im

    # take the median of each of the n values for each pixel
    med_frame = np.median(fits_stack, axis = 2)
    #write to new file
    fits.writeto(newname + ".fits", med_frame, biasheader, overwrite=True)
    print('Files median combined and saved')
    return med_frame

#define bias correction & subtraction function
def bias_correction(filelist, path_to_bias):
    """
    This function takes in a list of images and a master bias image and calculates the corrections to the bias using the
    overscan of each image. It then bias subtracts each image with this new, corrected master bias and saves the images
    with the prefix b_.
    """
    n = len(filelist)
    for ii in range(n):
        #getting data from images
        im_data = fits.getdata(filelist[ii])

        new_header = fits.getheader(filelist[ii])

        bias_data = fits.getdata(path_to_bias)

        #isolating the overscan from the images
        overscan = im_data[4100:4140,4100:4140]

        #take median of overscan region and master bias
        osmed = np.median(overscan)

        MBmed = np.median(bias_data)

        #make new master bias by normalizing the old one and scaling it by the overscan median
        MBnew = bias_data/MBmed

        MBnew = MBnew*osmed
```

```
        #subtract new bias from image
        subtract_data = im_data - MBnew

        #delete overscan now that we are done with it
        new_image = subtract_data[0:4096,0:4096]

        #save as new fits file
        fits.writeto('b_' + filelist[ii],new_image, new_header, overwrite = True )
    print('Files bias subtracted and saved with prefix b_')
    return


def dark_subtract(filelist, path_to_dark):
    '''
    takes a fits file and a path to dark files as inputs and returns the fits file
    subtracted by the dark of corresponding exposure time
    '''
    n = len(filelist)

    for ii in range(n):
        header = fits.getheader(filelist[ii])
        frame = fits.getdata(filelist[ii])
        frame_exp = header['EXPTIME']
        if (frame_exp>=1):
            darkfile = path_to_dark + str(int(frame_exp)) + '.0s'+ '/Master_Dark_' + str(int(frame_exp)) + '.0s.fit'
        else:
            darkfile = path_to_dark + str(frame_exp) + 's'+ '/Master_Dark_' + str(frame_exp) + 's.fit'
        dark_header = fits.getheader(darkfile)
        dark_frame = fits.getdata(darkfile)
        subtract_frame = frame - dark_frame

        fits.writeto('d'+ filelist[ii], subtract_frame, header, overwrite=True)
    print('Files dark subtracted and saved with prefix d_')
    return

def norm_combine_flats(filelist, filtername):
    '''
    function that takes a list of fits files as inputs and returns an array of the median of the median-normalized images
    '''
    #NEEDS TO TAKE ONLY ONE FILTER
    # get the number of files and put into variable n
    n = len(filelist)

    flat_header = fits.getheader(filelist[0])

    # get data of first file in list
    first_frame_data = fits.getdata(filelist[0])

    # get size of first file in list
    imsize_y, imsize_x = first_frame_data.shape

    # create empty 3D array where each slice is the size of each file and axis 2 contains n files
    fits_stack = np.zeros((imsize_y, imsize_x , n))

    # loop over axis 2 and fill with normalized image values
    for ii in range(0, n):
        im = fits.getdata(filelist[ii])
        norm_im =  im/np.median(im)
        fits_stack[:,:,ii] = norm_im

    # take the pixel-wise median between all the normalized files and return the resulting array
    med_frame = np.median(fits_stack, axis=2)

    fits.writeto('Master_Flat_' + filtername +'.fits', med_frame, flat_header, overwrite=True)
    print('Files norm combined and saved as Master_Flat_' + filtername + '.fit')
```

```python
    return med_frame

def flat_divide(filelist, path_to_flat):
    '''
    takes file and divides master flatfield from it
    '''
    n = len(filelist)
    for ii in range(n):
        frame_data = fits.getdata(filelist[ii])

        header = fits.getheader(filelist[ii])

        master_flat = fits.getdata(path_to_flat)

        divide_data = frame_data/master_flat

        fits.writeto('F_' + filelist[ii], divide_data, header,overwrite = True)
        print('Files flat divided and saved with prefix F_')
    return

def centroid(image, bgy1, bgy2, bgx1, bgx2, sty1, sty2, stx1, stx2):
    '''
    function takes an image, the area of the image which will be used for background calculations, and the area which
    contains a star we will find the centroid of. calculates the centroid, outputting x and y coordinates
    '''
    sci = image[sty1:sty2,stx1:stx2]        #region of image
    bg_data = data[bgy1:bgy2, bgx1:bgx2]    #background of image
    bg_st = np.std(bg_data)   #standard dev of background
    sub_data = sci - 3*bg_st  #subtract 3stdev background from science frame to find relevant pixels
    xsum = np.array([])       #empty arrays for the x and y sums
    ysum = np.array([])

    plt.imshow(sci)
    #for loop goes through each value of x and y to find the numerator of the centroid eq.
    #then puts that information into an array
    for x in np.arange(np.shape(sci)[1]):
        xnum = np.sum(sub_data[:,x])*x
        xsum = np.append(xsum, xnum)
    for y in np.arange(np.shape(sci)[0]):
        ynum = np.sum(sub_data[y,:])*y
        ysum = np.append(ysum, ynum)


    #then we sum the new array values
    xnum_tot = np.sum(xsum)
    ynum_tot = np.sum(ysum)
    den = np.sum(sub_data)

    #then we put the equation together using the components we just calculated:
    xcentroid = xnum_tot/den
    ycentroid = ynum_tot/den

    plt.plot(xcentroid,ycentroid,'ro')
    print(xcentroid,ycentroid)
    return xcentroid, ycentroid

def shift_calc(imlist):
    n = len(imlist)
    #arrays to be filled; the x and y location of the centroids of each image, and the x and y shifts
    xcent = ([])
    ycent = ([])
    xshift = ([])
    yshift = ([])

    #loop thru each image
    for ii in range(n):
```

```
        #get data of each image
        image_file = imlist[ii]
        image_data = fits.getdata(image_file, ext=0)
        data = np.asarray(image_data)

        #use centroid function
        xcentroid,ycentroid = centroid(image_data, 3180,3250,660,950, 3030,3250,660,950)

        #put outputs of x and y centroid location into array
        xcent = np.append(xcent, xcentroid)
        ycent = np.append(ycent, ycentroid)

        #calculate xy shifts and append into array
        x = xcent[0] - xcent[ii]
        xshift = np.append(xshift,x)
        y = ycent[0] - ycent[ii]
        yshift = np.append(yshift,y)
    print('xshift and yshift computed')
    return xshift, yshift

# make a function that takes offsets and pads images
def align(imlist,xshift,yshift,pixel):
    '''
    function that takes a list of images which have been through the centroid function i.e. for which we have calculated
    shifts in x and y direction, and a pixel value, and outputs shifted images padded by the pixel value so they are
    aligned.
    '''
    n = len(imlist)

    for ii in range(n):
        #get image data
        header = fits.getheader(imlist[ii])
        file = fits.getdata(imlist[ii])
        filename = imlist[ii]

        #pad each file with the given pixel number
        new_array = np.pad(file, pixel,'constant', constant_values = -0.001)

        #if statement to make sure we dont shift the image if there is no shift
        if (xshift[ii]&yshift[ii]==0):
            new_array2 = new_array

        else:
            new_array2 = interp.shift(new_array, (xshift[ii],yshift[ii]),cval = -0.001)
        new_array2[new_array2 <=-0.0001] = np.nan

        #write to new file
        fits.writeto('pad_' + filename, new_array2, header, overwrite = True)
    print('Files aligned and saved with prefix pad_')
    return new_array2

def filesorter(filename, foldername, fitskeyword_to_check, keyword):
    '''
    This function takes input file and places it in a folder given if it is the correct type of fits file.
    '''
    # this checks if there is a file of that name already, if so then it moves onto the next step, otherwise
    # tells you it does not exist.
    if os.path.exists(filename):
        pass
    else:
        print(filename + " does not exist or has already been moved.")
        return

    header = fits.getheader(filename)
    fits_type = header[keyword]
```

```python
    # this checks that there is a folder of that name, and makes it if there is not
    if os.path.exists(foldername):
        pass
    else:
        print("Making new directory: " + foldername)
        os.mkdir(foldername)

    # this checks that the file is the correct type of fits file and moves it to the folder if it is
    if fits_type == fitskeyword_to_check:
        destination = foldername + '/'
        print("Moving " + filename + " to: ./" + destination + filename)
        os.rename(filename, destination + filename)
    return

def badpixelcorrect(data_arr, badpixelmask, speed = 'fast'):
    '''
    badpixelcorrect
    -----------------
    Performs a simple bad pixel correction, replacing bad pixels with image median.
    Input image and bad pixel mask image must be the same image dimension.

    inputs
    ----------------
    data_arr      : (matrix of floats) input image
    badpixelmask  : (matrix of floats) mask of values 1.0 or 0.0, where 1.0 corresponds
                    to a bad pixel
    speed         : (str) whether to calculate the median filtered image (computationally
                    intensive), or simply take the median of the full image. Default = 'fast'

    outputs
    ---------------
    corr_data     : (matrix of floats) image corrected for bad pixels

    '''
    corr_data = data_arr.copy()

    if speed == 'slow':
        # smooth the science image by a median filter to generate replacement pixels
        median_data = ndimage.median_filter(data_arr, size=(30,30))

        # replace the bad pixels with median of the local 30 pixels
        corr_data[badpixelmask == 1] = median_data[badpixelmask == 1]

    else:
        corr_data[badpixelmask == 1] = np.nanmedian(data_arr)

    return corr_data
```