```
import numpy as np
import sep
import matplotlib.pyplot as plt
from matplotlib.patches import Ellipse
from astropy.visualization import ZScaleInterval,ImageNormalize
from astropy.table import Table

gain = 0.3360284075579739 # What you found in Assignment 2
read_noise_electrons = 1.6506094506539937 # What you found in Assignment 2
# Max ADU Linear region
saturation_level = 64691  # What you found with linearity measurements in Ass
median_dark_current_electrons = 0.000747 # From Assignment 2

#-------------------------------------------------------------------------
def findObjects(image,binning=1,edge_dist=30,F=3,B=64,brightest=None,nonblend
    '''
    Given image data, this function returns an astropy table of objects found
    image using sep (Source Extractor python), sorted from brightest to dimme
    rows are the objects, and the columns are the properties.

    image should be provided in ADU/s to properly account for the dark noise

    Objects are excluded from the output if they are within edge_dist pixels
    edges of the image.

    If brightest is set, keep only the N brightest objects.  Eg. if brightest
    only the 20 brightest objects.

    nonblended = True means keep only on the stars/objects that are not close
    star.  Adjacent stars will often overlap in their fluxes, so that fluxes
    "blended".  (More about that later.)
    '''

    # If SEP produces an error, replace the following line with: image_data =
    #image_data = image.byteswap(inplace=False).newbyteorder() ## Produced an
    image_data = image

    # determine the sky level as a function of position
    sky = sep.Background(image_data,fw=F,fh=F,bh=B,bw=B)
    sky_data = sky.back() # This is the sky level in each pixel (ADUs)
    image_data_nosky = image_data - sky_data # Subtract the sky data from the

    #-------------------------------------------------------------------------
    # Calculate the total noise properly
```

```
# The random noise in the sky-subtracted image_data_nosky is a combinatio
# quadrature) of the read noise and the Poisson noise.  The Poisson noise
# the Poisson of the sky, the Poisson noise of the dark current, and the
# noise of the image.  Compute the total random noise taking these into a

# Poisson Noise sqrt(intensity (or counts) in (ADUs). * Gain)
sky_noise_electrons = np.sqrt(sky_data*gain) # The Poisson noise of the s
image_noise_electrons = np.sqrt(image_data*gain) # The Poisson noise of t
                                                 # image in electrons
# The Poisson noise of the dark signal in electrons
dark_noise_electrons = np.sqrt(median_dark_current_electrons*gain)


# Compute the read noise (in electrons) appropriate for the binning used.
# noise you found in Assignment 2 was for the original unbinned pixels.
# binning, the read noise for the "superpixel" is the read noise for the
# pixels added in quadrature.  Do the same for the dark noise.

read_noise_superpixel_electrons = np.sqrt(read_noise_electrons * 4)
dark_noise_superpixel_electrons = \
    np.sqrt(median_dark_current_electrons * 4 * texp)

# Add all the noise (in electrons) in quadrature
noise_electrons = np.sqrt(sky_noise_electrons**2 +
                          image_noise_electrons**2 +
                          read_noise_superpixel_electrons**2 +
                          dark_noise_superpixel_electrons**2
)
noise = noise_electrons / gain  # Convert back to ADUs # Compute the nois
#"noise" will be used in the call to sep.sum_ellipse() below in order to
# proper errors in the flux measurements

#-----------------------------------------------------------------------
# This line extracts all light sources at least 1.5 sigma from the backgr
# (sky.globalrms) and assigns the result to the variable called "objects"
# We went over in class how to do this.
objects = sep.extract(image_data_nosky, 1.5, err=sky.globalrms)
objects = Table(objects) # turn into an astropy Table object

#-----------------------------------------------------------------------
# These lines trim remove bad objects from the "objects" table
## Type objects in the terminal to look at the table easily
# Remove saturated objects.  Check the value of objects['peak'] which is
```

```
# value of the brightest pixel included in the object
mask = objects['peak'] < 60000  # True for non-saturated objects
objects = objects[mask]

# Exclude objects within edge_dist pixels of the edges of the image
# Use objects['x'] and objects['y']
mask = ((objects['y'] > edge_dist) & \
        (objects['y'] < image_data.shape[0] - edge_dist)) \
    &   ((objects['x'] > edge_dist) & \
         (objects['x'] < image_data.shape[1] - edge_dist))

objects = objects[mask]

# Remove objects that are "blended" or too near other objects (written fo
if nonblended:
    mask = (objects['flag'] == 0) # see Source Extractor documentation
    objects = objects[mask]

# Sort the objects table from brightest to dimmest (using 'flux' which is
# estimate of the flux)  (written for you)
objects.sort('flux') # dimmest to brightest
objects = objects[::-1] # brightest to dimmest

# If the "brightest" keyword is set, keep only the N brightest objects.
# brightest is 20, keep the 20 brightest objects. (written for you)
if brightest: objects = objects[0:int(brightest)]

# End of section that trims objects
#-----------------------------------------------------------------------
# The rest of this function has to do with standard definitions of radii
# and they are already completed for you (we will learn more about these
# another time).

# define some short-hand names for the columns
x,y,a,b,theta = objects['x'],objects['y'],objects['a'], \
    objects['b'],objects['theta']

# theta should be between -pi/2 and pi/2, but due to numerical floating p
# errors, sometimes theta is beyond this range.  If that is the case, add
# subtract pi
mask = (theta > np.pi/2)
theta[mask] -= np.pi
mask = (theta < -np.pi/2)
theta[mask] += np.pi
```

```
    # This line computes the Kron radius of each object (in units of a and b)
    kronrad, krflag = sep.kron_radius(image_data_nosky, x,y,a,b,theta,6.0)

    # This line measures the flux within 2.5*Kron radius
    flux, fluxerr, flag = sep.sum_ellipse(image_data_nosky, x,y,a,b,theta,
                                          2.5*kronrad,subpix=1,err=noise)

    # In some cases the kronrad is too small.  Impose a minimum and use circu
    # apertures for those objects
    r_min = 1.75  # minimum diameter = 3.5
    mask = (2.5*kronrad * np.sqrt(a * b) < r_min)
    cflux, cfluxerr, cflag = sep.sum_circle(image_data_nosky, x[mask], y[mask
                                            r_min, subpix=1)
    flux[mask] = cflux
    fluxerr[mask] = cfluxerr
    flag[mask] = cflag

    flux_radius, flag = sep.flux_radius(image_data_nosky, x,y,6.*a,
                                        0.5,normflux=flux,subpix=5)

    sig = 2. / 2.35 * flux_radius
    xwin, ywin, flag = sep.winpos(image_data_nosky,x,y, sig)

    # Add some useful quantities to the objects table
    objects['flux_auto'] = flux  # A more accurate measure of flux than "flux
    objects['flux_auto_err'] = fluxerr
    objects['flux_radius'] = flux_radius # The radius that contains half the
    objects['xwin_image'] = xwin # A more accurate centroid than "x"
    objects['ywin_image'] = ywin # A more accurate centroid than "y"



    return objects,image_data_nosky

#-----------------------------------------------------------------------------
def plotObjects(objects,image,k=5):
    # scales the image by same 'zscale' algorithm as ds9
    fig, ax = plt.subplots()
    norm = ImageNormalize(image,interval=ZScaleInterval())
    ax.imshow(image, cmap='gray',origin='lower',norm=norm)
    # define some arrays
    x,y,a,b,theta = objects['x'],objects['y'],objects['a'],\
        objects['b'],objects['theta']
```

```
    # Plot an ellipse for each object.  Look up the documention for Ellipse a
    # appropriate columns from objects to fill in the parameters for Ellipse.
    # want to muliply the semimajor/minor axes by some factor so that the ell
    # be big enough to see.

    ## a is semi-Major Axis, b is the Semi-Minor Axis
    ## constant is k


    for i in range(len(objects)): # Loop through each object
        e = Ellipse(xy=(x[i],y[i]),
                    width= k*a[i],
                    height= k*b[i],
                    angle= theta[i])
        # note that theta is in radians, but angle needs to be in degrees
        e.set_facecolor('none')
        e.set_edgecolor('red')
        ax.add_artist(e)
    return fig,ax
#-------------------------------------------------------------------------
```