Photometry of processed CCD images

In this notebook, we'll go over how to measure the amount of light we received from a star.

Measuring the amount of received light is known as *photometry*.

Learning goals are:

- Learn to measure a star's light from a CCD image
- Gain further insight into uncertainty and error propagation

Index

In [1]:
```
%matplotlib inline
```

In [2]:
```
import numpy as np
import pylab as pl
pl.rcParams['image.origin'] = 'lower' # make images display right-side-up
pl.style.use('dark_background')
```

Step 1: Load the data

We want to perform photometry on *reduced*, fully-processed data. We'll start by loading up the data we processed as part of the single-frame reduction exercise.

In [3]:
```
from astropy.io import fits
import os
```

In [4]:
```
# on my computer, the data are in a subdirectory called 'data/', so I chdir there
#os.chdir('data/')
```

In [5]:
```
pleiades_clear_darksub_flattened = fits.getdata('pleiades_30s_C_other_001_darksub
pleiades_clear_darksub_flattened_uncertainty = fits.getdata('pleiades_30s_C_other
```

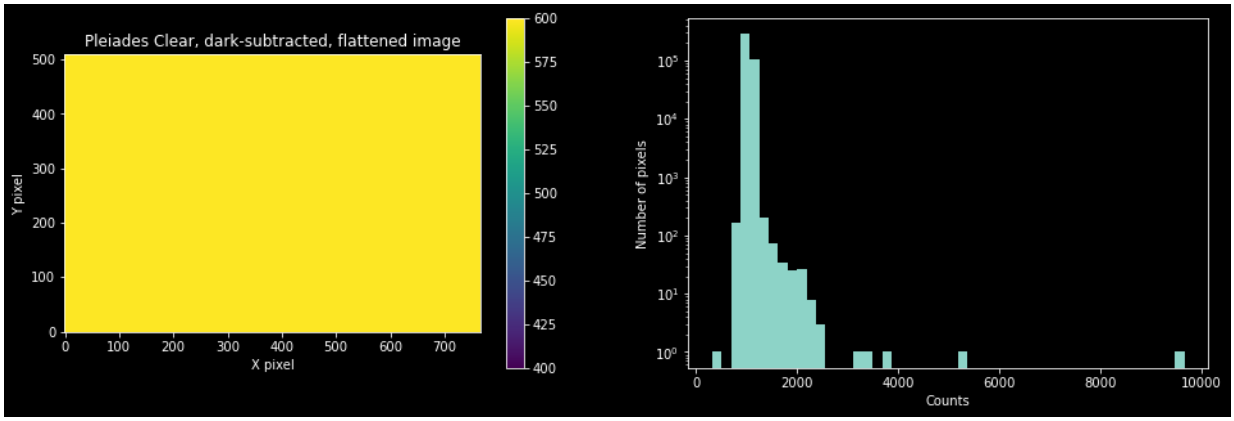
Quick consistency check: Make sure these numbers match up. If either of these numbers are not correct, you need to go back and re-run the Data Reduction - Single Frame notebook.

In [6]:
```
print(f"This value: {pleiades_clear_darksub_flattened_uncertainty[50,50]:0.2f} sh
print(f"This value: {pleiades_clear_darksub_flattened[50,50]:0.2f} should be 952.
```

```
This value: 39.72 should be 39.72
This value: 952.24 should be 952.24
```

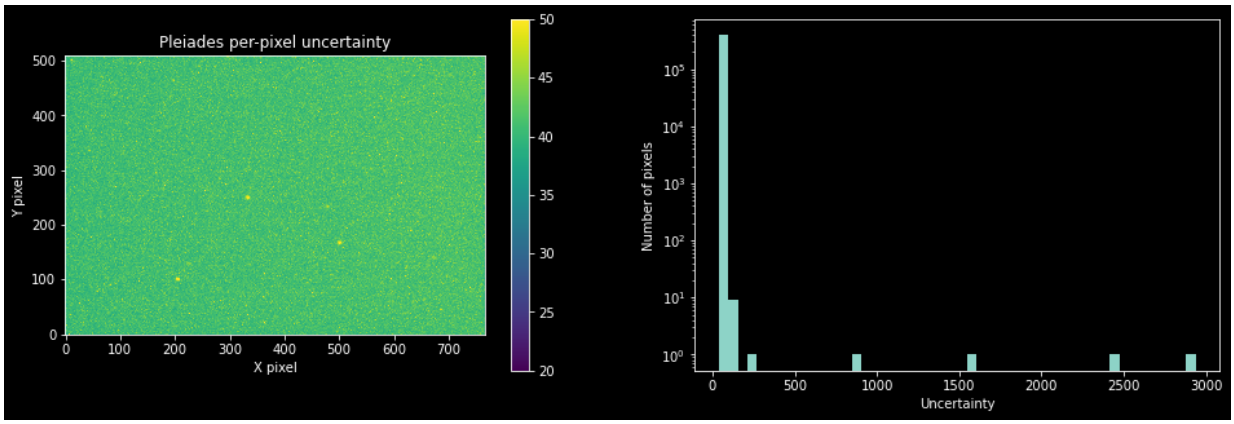
As usual, the first thing we do is display & inspect the data

In [7]:
```
pl.figure(figsize=(16,5))
im = pl.subplot(1,2,1).imshow(pleiades_clear_darksub_flattened, origin='lower', i
pl.colorbar(im)
_=pl.title("Pleiades Clear, dark-subtracted, flattened image")
_=pl.xlabel("X pixel")
_=pl.ylabel("Y pixel")

_ = pl.subplot(1,2,2).hist(pleiades_clear_darksub_flattened.ravel(), bins=50, log
_ = pl.xlabel("Counts")
_ = pl.ylabel("Number of pixels")
```



And the uncertainty:

In [8]:
```
pl.figure(figsize=(16,5))
im = pl.subplot(1,2,1).imshow(pleiades_clear_darksub_flattened_uncertainty,
                              origin='lower', interpolation='none', vmax=50, vmin
pl.colorbar(im)
_=pl.title("Pleiades per-pixel uncertainty")
_=pl.xlabel("X pixel")
_=pl.ylabel("Y pixel")

_ = pl.subplot(1,2,2).hist(pleiades_clear_darksub_flattened_uncertainty.ravel(),
_ = pl.xlabel("Uncertainty")
_ = pl.ylabel("Number of pixels")
```



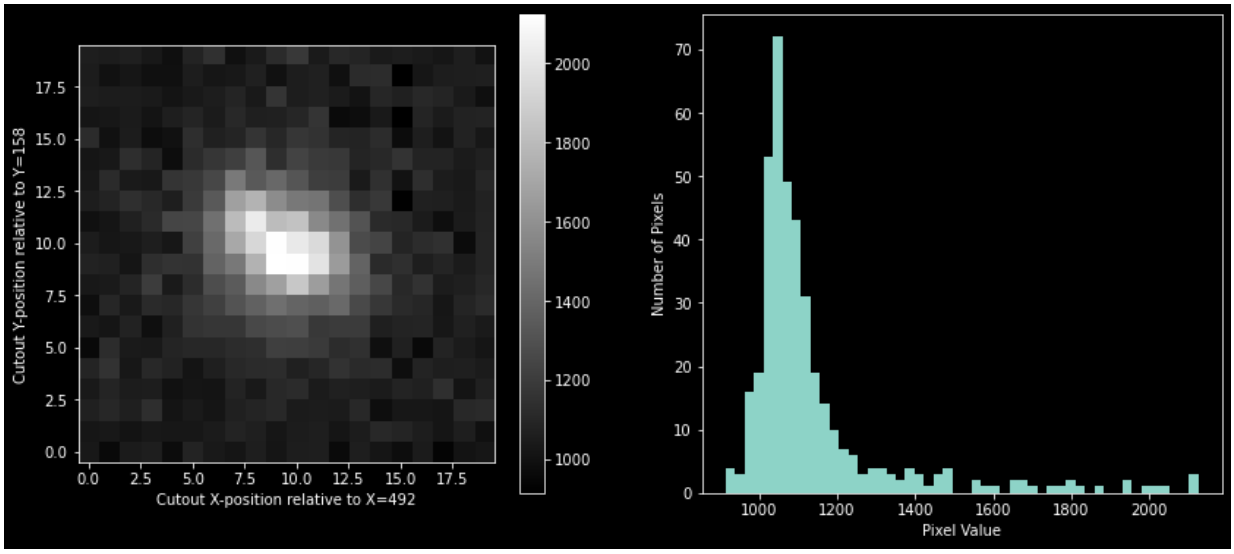
Photometry

Our next step is to perform some measurements of the stars.

To measure the light from a single star, we need to select only pixels containing that star's light.

As a first attempt, we can just select a small box containing the star. You can figure out the size, shape, and location of this box by eyeballing the image above, then iterating a few times to get the right spot. In other words, you might have started with a box $\sim$ 30 pixels on a side centered on $x \sim 500, y \sim 175$, which would be `image[175-15:175+15, 500-15:500+15]`, then gradually changed the center and limits, each time inspecting the image, until you have a tightly-framed box around the star, as below:

In [9]:
```
cutout_star = pleiades_clear_darksub_flattened[158:178, 492:512]
pl.figure(figsize=(14,6))
im = pl.subplot(1,2,1).imshow(cutout_star, cmap='gray', origin='lower')
pl.xlabel("Cutout X-position relative to X=492")
pl.ylabel("Cutout Y-position relative to Y=158")
pl.colorbar(im)
_=pl.subplot(1,2,2).hist(cutout_star.ravel(), bins=50)
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



The image shows our star, which is a little distorted. The distortion is most likely from *tracking errors* in the telescope: it is not rotating exactly correctly to follow the sky.

You should also note that the floor of the image, the lowest observed flux measurement, is not zero. The floor is closer to ~500. This background is most likely produced by sky contamination. From CTO, that most likely means scattered light from Gainesville.

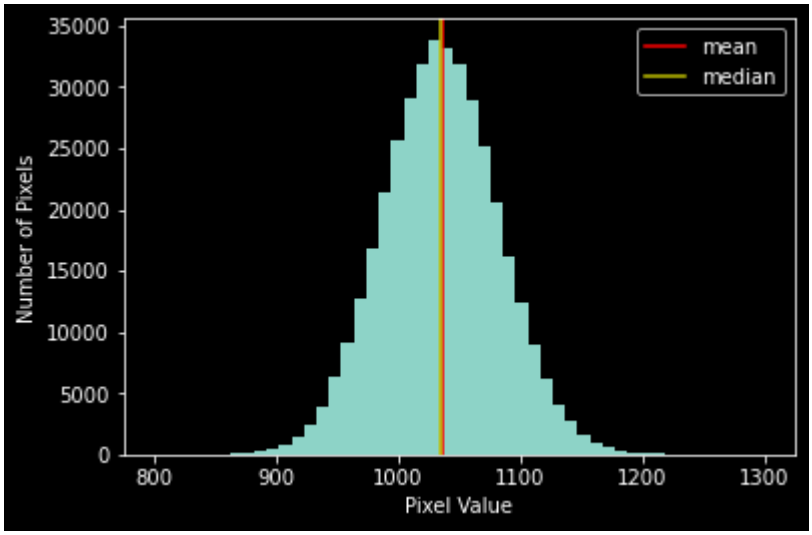
To remove this background, we need to obtain an estimate of the sky value.

Because the image is almost entirely sky pixels (the target stars only take up a very small number of them), we can assume that the whole image is effectively an image of the sky background. We can therefore use the mean (or, more robustly, the median) of the whole image as our sky estimate (but note that this is not possible for all observations - for images of M42, for example, the nebula fills the CCD, so you need separate images to estimate the sky contamination):

In [10]:
```
sky_mean = pleiades_clear_darksub_flattened.mean()
sky_median = np.median(pleiades_clear_darksub_flattened)
print(sky_mean, sky_median)
```

```
1035.2701922573067 1034.4351460591993
```

In [11]:
```
_=pl.hist(pleiades_clear_darksub_flattened.ravel(), bins=np.linspace(800,1300))
_=pl.axvline(sky_mean, color='r', label='mean')
_=pl.axvline(sky_median, color='y', label='median')
_=pl.legend(loc='best')
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



These values are nThese values are nearly the same, so we can use either safely. If the mean and median didn't line up, we would then look for a source of bias - maybe there are outliers that affect the mean, for example.early the same, so we can use either safely.

Q1

Index

Let's briefly take an aside - what is the uncertainty on this measurement of the sky background? Recall that the error on the mean, σ_μ, is the uncertainty on any individual measurement divided by the number of measurements, i.e.: $\sigma_\mu = \sigma_i/\sqrt{N}$.

Each individual pixel's value is an independent measurement of the sky.

$$\Sigma_{sky} = \sum x_i$$

The propagation of error formula for sums gives us the error on the sum.

$$\sigma_\Sigma^2 = \sum \sigma_i^2 = N\sigma_i^2$$

If we assume σ_i is constant (every pixel has the same uncertainty), we can simplify this further:

$$\sigma_\Sigma^2 = \sum \sigma_i^2 = N\sigma_i^2$$

so

$$\sigma_\Sigma = \sqrt{N}\sigma_i$$

We want to calculate the \emph{mean}, which is defined as the sum divided by the number of elements summed over, i.e.:

$$\mu_{sky} = \frac{1}{N}\sum x_i$$

The propagation of error formula gives us, again:

$$\sigma_\mu^2 = \frac{1}{N^2}\Sigma\sigma_i^2$$

So we have:

$$\sigma_\mu = \left(\frac{1}{N^2}\Sigma\sigma_i^2\right)^{1/2} = \frac{1}{N}\sigma_\Sigma$$

which, if we put in σ_Σ from above, gets us to:

$$\sigma_\mu = \frac{1}{N}\sigma_\Sigma = \frac{\sqrt{N}\sigma_i}{N} = \frac{\sigma_i}{\sqrt{N}}$$

We calculated the per-pixel uncertainty, and loaded it into this notebook, from the previous exercise using our usual propagation of error formulae.

In [12]:
```
# first, let's calculate the error on the _sum_ of all sky pixels
uncertainty_sky_sum = (pleiades_clear_darksub_flattened_uncertainty**2).sum()**0.
print(f"uncertainty_sky_sum={uncertainty_sky_sum}  (should be 26166.656)")
```

```
uncertainty_sky_sum=26166.65562228448  (should be 26166.656)
```

We have a lot of pixels. N is:

In [13]:
```
pleiades_clear_darksub_flattened.size
```

Out[13]:
```
391170
```

In [14]:
```
# given the above, what is the uncertainty on the _mean_ sky estimate?
uncertainty_on_sky_mean = uncertainty_sky_sum/(pleiades_clear_darksub_flattened.s
uncertainty_on_sky_mean
```

Out[14]:
```
0.06689330884854279
```

In [15]:
```
print(f"The sky mean is {sky_mean:0.2f} +/- {uncertainty_on_sky_mean:0.2f}.  The
```

```
The sky mean is 1035.27 +/- 0.07.  The SNR=1.5e+04
```

You should find the uncertainty is very small (~0.04-0.05), such that the S/N of our "sky background" measurement is ~10,000.

" The sky mean is 464.25 +/- 0.05. The SNR=8.4e+03 "

Note that there are two ways to compute the per-pixel uncertainty: you can follow the error propagation rules we used above or, in cases where you have a lot of pixels independently measuring the same signal (like this case), you can directly estimate the noise from the standard deviation of the data:

In [16]:
```
sky_stddev = pleiades_clear_darksub_flattened.std() # estimate of the per-pixel e
print(f"Standard deviation of the data: {sky_stddev:0.1f}")
print(f"Average of the per-pixel uncertainty from error propagation: {pleiades_cl
```

```
Standard deviation of the data: 53.1
Average of the per-pixel uncertainty from error propagation: 41.3
```

The per-pixel error is substantial, ~40 and this error will play an important role in our photometry measurements below.

Q2

Index

Another aside, going back to the atmosphere lecture: What color will the "sky background" be, and why? By "color" here, we mean the wavelength of light we measure, and particularly the relative amount of light at different wavelengths. Credit here is for explaining your reasoning - there are multiple possible correct answers, and we don't know which is correct without measuring. Bonus points if you come up with multiple mechanisms and can describe a measurement we can perform with our instruments to distinguish between them.

STUDENT ANSWER

The background color should be a darker blue due mostly to Rayleigh scattering in the atmosphere. There are some small effects that contribute as well such as light pollution, air glow, and zodical light. If you're close to moon, there is mie scattering which is less dependent on wavelength. This gives the halos we see around the moon, street lights, and some stars.

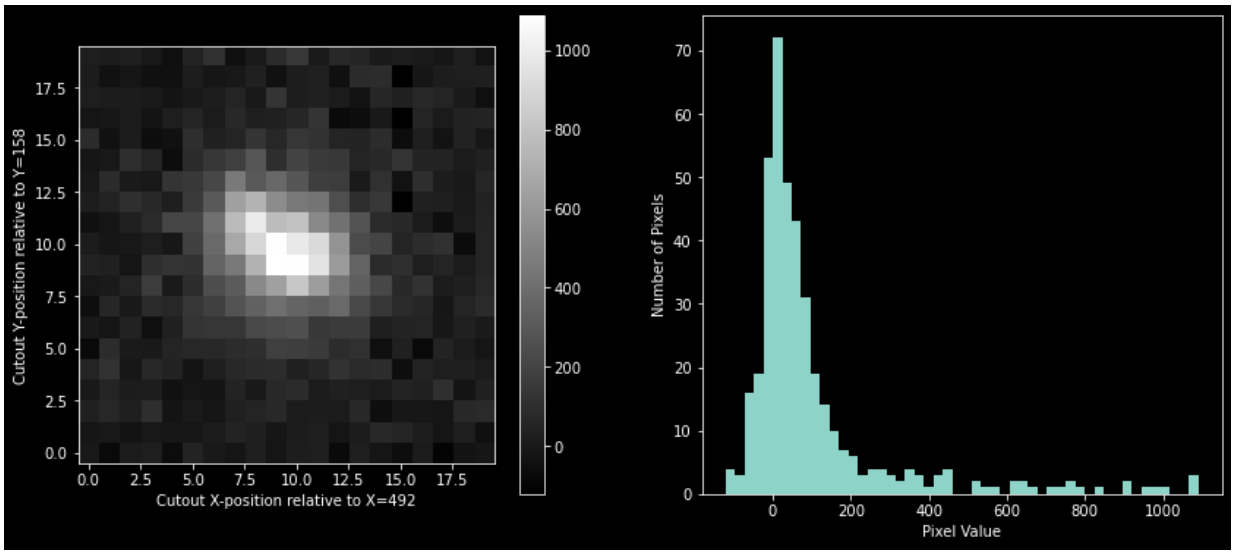
(back on-topic)

Now, we can subtract that sky value from the image:

In [17]:
```
sky_subtracted_pleiades_clear = pleiades_clear_darksub_flattened - sky_mean
```

Now we can look at it again, and we see that the background level is approximately zero:

In [172]:
```
cutout_star = sky_subtracted_pleiades_clear[158:178, 492:512]

pl.figure(figsize=(14,6))
im = pl.subplot(1,2,1).imshow(cutout_star, cmap='gray', origin='lower')
pl.xlabel("Cutout X-position relative to X=492")
pl.ylabel("Cutout Y-position relative to Y=158")
pl.colorbar(im)
_=pl.subplot(1,2,2).hist(cutout_star.ravel(), bins=50)
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



Note that the peak of the histogram is near (but not exactly) zero now that we have subtracted off the sky background. Ideally, it would be exactly zero; the remainder may just be noise or may be scattered light from this star or nearby stars.

So, how much flux is in the star? We can simply sum over the cutout area we've made:

In [19]:
```
star_sum = cutout_star.sum()
print(star_sum)
```

```
41349.20439525999
```

We have a flux measurement! We now need to measure the uncertainty on that measurement.

We can start by noting how many pixels we summed over:

In [20]:
```
cutout_star.size
```

Out[20]:
```
400
```

The error per pixel is, to first order, the same as the error per pixel of the sky background measurement. There's actually more noise than that because of the higher counts in the star (which increases Poisson noise), but let's start with the sky-based noise estimate.

What contributes to the sky noise σ_{sky}? At least three components:

- read noise
- Poisson noise from dark current
- Poisson noise from the sky.

However, we don't need to know the value of each component, we just need to know the per-pixel uncertainty σ_{sky}

What is the error on our measurement, then? It's the error on the sum, which is the sum in quadrature of the errors on the individual measurements. As long as the error on each pixel is the same, we have:

$$\sigma_{sum} = \left(\Sigma_i \sigma_i^2\right)^{1/2} = \left(N\sigma_i^2\right)^{1/2} = N^{1/2}\sigma_i$$

In [21]:
```
star_sum_error = (cutout_star.size * sky_stddev**2)**0.5
star_sum_error
```

Out[21]:
```
1062.780308450075
```

Starting with just this source of error, what's our signal-to-noise ratio?

In [22]:
```
star_sum / star_sum_error
```

Out[22]:
```
38.906633917184976
```

That's a signal-to-noise ratio of about 40.

Q3

Index

For a signal-to-noise ratio of 40, what is the *approximate* magnitude error?

Remember the magnitude error is approximately the inverse of the signal-to-noise ratio for small enough uncertainties.

In [23]:
```
# student answer
mag_err =  1/40
mag_err
```

Out[23]:
```
0.025
```

That estimate ignored the Poisson noise from the star.

Adding Poisson error from the star to the Sky uncertainty

To more accurately determine the error, which is a little larger than that value above, we need to account for the Poisson noise from the star itself.

Recall that the total noise in a pixel has several contributions:

$$\sigma^2_{pixel} = \sigma^2_{readnoise} + \sigma^2_{dark,Poisson} + \sigma^2_{sky,Poisson} + \sigma^2_{star,Poisson}$$

We noted before that the sky noise σ_{sky} contains contributions from dark, readnoise, and sky, and we have measured it. We assume that the background noise (sky, dark, and readnoise) is uniform (doesn't change anywhere in our image). So, given

$$\sigma^2_{sky} = \sigma^2_{readnoise} + \sigma^2_{dark,Poisson} + \sigma^2_{sky,Poisson}$$

we can substitute our measured σ_{sky} in the first equation above to get:

$$\sigma^2_{pixel} = \sigma^2_{sky} + \sigma^2_{star,Poisson}$$

Since we've already measured σ_{sky}, we only need to measure $\sigma_{star,Poisson}$, which we can obtain from $\sqrt{\lambda}$, i.e., the square root of the number of counts. In other words, $\sigma^2_{star,Poisson} = n_{counts}$

Recall that the `cutout_star` image should now consist only of starlight measurement, with no contribution from dark, bias, or sky background since we subtracted off the bias and sky background previously. That means that each pixel is a measurement of the number of counts (if you account for the gain), which is the squared error.

The operation below measures the error on *each pixel*:

In [24]:
```
# we re-load the gain as a reminder, but if we did the dark subtraction and flatt
gain = fits.getheader('pleiades_30s_C_other_001_darksub_flattened.fits')['EGAIN']
gain
```

Out[24]: 2.23

In [25]:
```
# we just rename the variable to make more clear that the "cutout" contains elect
per_pixel_electron_counts_star_cutout = cutout_star
```

In [26]:
```
per_pixel_error = (per_pixel_electron_counts_star_cutout + sky_stddev**2)**0.5
```

So, now what is the true total error? Again we simply sum in quadrature ($\sigma^2_\Sigma = \Sigma\sigma^2_i$):

In [27]:
```
total_star_sum_error = (per_pixel_error**2).sum()**0.5
total_star_sum_error
```

Out[27]: 1082.0587730915988

That error is only a little larger than the previous one.

Noting the *peak counts* (the brightest pixel) in the star are ~1000:

In [28]:
```
cutout_star.max()
```

Out[28]:
```
1089.3864524679968
```

Q4

Index

Explain why the error when including the Poisson noise from the star is only a little larger than the error when *not* including it.

Hint: How many photons did you get from the star in the brightest pixel? How many photons did you get from the sky (background) in that same pixel?

STUDENT ANSWER

The peak counts for the brightest pixel are 1089, the counts for the whole sky as a mean are 1035.2701922573067. This number isn't very different from the brightest pixel. We get some number of photons both from the sky and star that contribute to the total uncertainty. sqrt(2000) vs sqrt(1000)

In the end, we have a measurement $S_* = 41300 \pm 1100$ with SNR = 38

In [29]:
```
# I used this print statement to get the above answer
# int() converts a float measurement to an integer - so instead of printing 41000
# np.around is a rounding tool.  I told the code to round each of the numbers to
# (we estimated that the error has two significant figures, but we matched the st
# of significant figures)
print(f"In the end, we have a measurement $S_* = {int(np.round(star_sum, -2))} \p
      f" with SNR = {int(np.around(star_sum / total_star_sum_error, 0))}")
```

```
In the end, we have a measurement $S_* = 41300 \pm 1100$ with SNR = 38
```

Optimizing our Measurement

Index

However, note that this is not an optimal measurement. We included *noise* from many pixels that did not contain any signal, thereby reducing our signal-to-noise ratio.

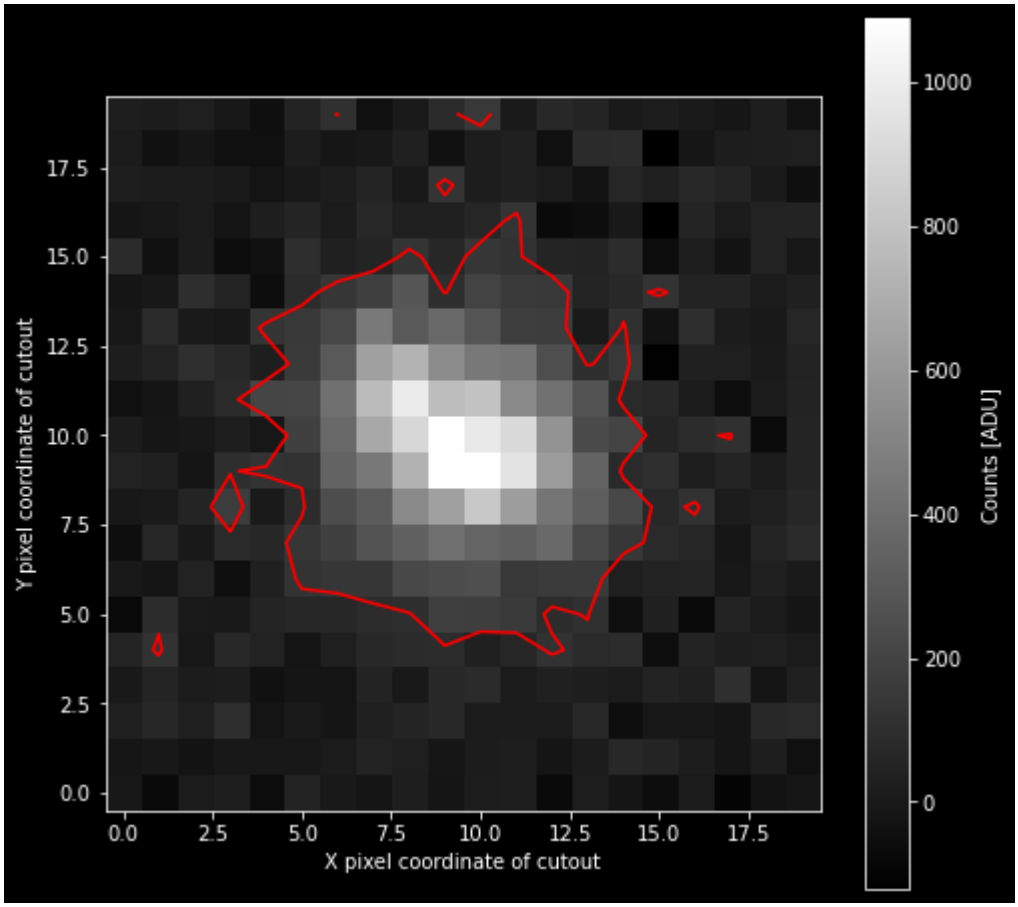
We can improve this measurement by excluding the pixels that only contribute noise and do not contribute signal.

To figure out which pixels these are, we can apply a signal-to-noise threshold.

We know the typical per-pixel noise is ~ 50 counts, so if we include only pixels $S > 2\sigma$ ($S > 100$ counts), we might see only the star. Note that 2σ is an arbitrary threshold - different thresholds (3σ, 5σ, etc) may be more appropriate in other situations.

In the next plot, we'll overlay *contours* selecting the region with $S > 2\sigma$:

In [30]:
```
pl.figure(figsize=(8,8))
im = pl.imshow(cutout_star, cmap='gray', origin='lower')
cb = pl.colorbar(im)
cb.set_label("Counts [ADU]")
pl.contour(cutout_star, levels=[2*sky_stddev], colors=['r'])
_=pl.xlabel("X pixel coordinate of cutout")
_=pl.ylabel("Y pixel coordinate of cutout")
```



We can create a *mask* to select only pixels above the threshold:

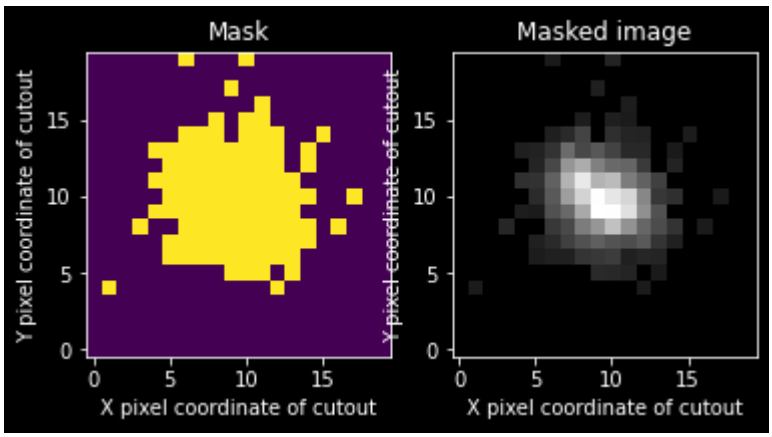
In [31]:
```
mask = cutout_star > 2*sky_stddev
```

The `mask` is a *boolean array mask*, in which each pixel is either `True` or `False`.

If you multiply another number by a boolean, like `5*True` or `3*False`, they are treated as `True=1`, `False=0`.

Both of these are shown in the next figure:

In [32]:
```
pl.subplot(1,2,1).imshow(mask)
pl.title("Mask")
_=pl.xlabel("X pixel coordinate of cutout")
_=pl.ylabel("Y pixel coordinate of cutout")
pl.subplot(1,2,2).imshow(cutout_star*mask, cmap='gray')
pl.title("Masked image")
_=pl.xlabel("X pixel coordinate of cutout")
_=pl.ylabel("Y pixel coordinate of cutout")
```



We can sum over only the included pixels now (those where `mask==True`), and compute the noise for only these pixels.

In [33]:
```
masked_star_sum = cutout_star[mask].sum()
print(f"Masked star sum = {masked_star_sum:0.1f}, "
      f"{-(masked_star_sum-star_sum)/star_sum*100:0.1f} percent less than {star_s
```

```
Masked star sum = 35316.2, 14.6 percent less than 41349.2
```

Since we summed over a smaller area, we also have a smaller contribution of sky noise. We repeat the total error calculation we made above, but now using only masked pixels. Note that `mask.sum()` is the number of pixels included in the mask:

In [34]:
```
masked_star_sum_error = (cutout_star[mask].sum() + sky_stddev**2*mask.sum())**0.5
print(masked_star_sum_error)
```

```
563.6414354273203
```

In [35]:
```
print(f"For the masked sum, we obtain $S_* = {int(np.round(masked_star_sum, -2))}
      f" with SNR = {int(np.around(masked_star_sum / masked_star_sum_error, 0))}"
```

```
For the masked sum, we obtain $S_* = 35300 \pm 600$ with SNR = 63
```

The new value for the sum is quite a bit different from the original, but it includes all the flux from the star.
The signal-to-noise ratio is higher because we didn't include as much noise.

Systematic Errors

Index

That extra 15% in the original measurement may come from several possible sources:

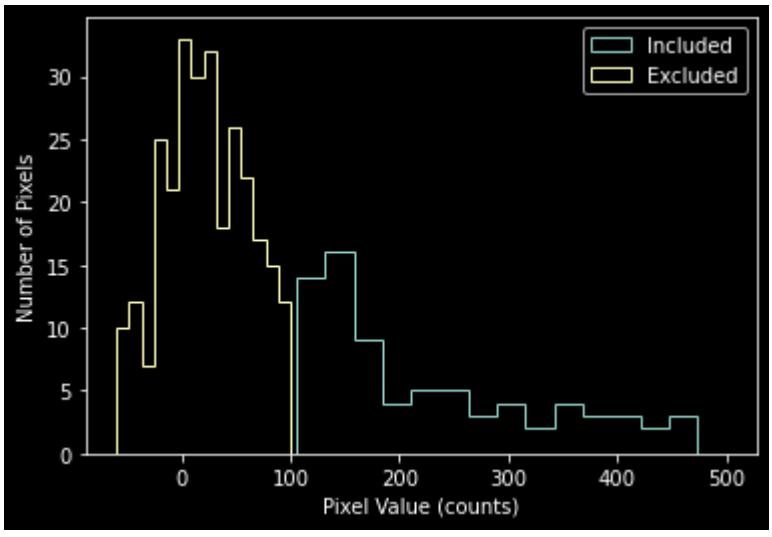
- Background stars that are too faint to see individually
- Localized sky emission that is different from the mean (this is unlikely to be significant)
- Localized nebular emission or reflected emission
- Inadequately-corrected flat-field effects
- Pixels containing some star flux that were excluded because they are below the threshold (this effect is often corrected using an *aperture correction*, which accounts for a well-understood point-spread-function [PSF])

Depending on how accurate we need our measurements to be, we may need to account for these errors. Flux loss errors like this are generally called *systematic errors* because they operate in only one direction (in this case, lower flux, not higher), as opposed to measurement errors that are *random* and tend to average to zero.

To search for systematic errors, we can examine our data more closely:

In [36]:
```
_=pl.hist(cutout_star[mask], bins=np.linspace(0,500,20), label='Included', histty
_=pl.hist(cutout_star[~mask], bins=np.linspace(-60,100,15), label='Excluded', his
_=pl.legend()
_=pl.xlabel("Pixel Value (counts)")
_=pl.ylabel("Number of Pixels")
print(f"Mean of 'background' region: {cutout_star[~mask].mean():0.1f}, "
      f"Sum of 'background' region: {cutout_star[~mask].sum():0.1f}")
```

```
Mean of 'background' region: 20.1, Sum of 'background' region: 6033.0
```



We can see from the above histogram and arithmetic that the background pixels have a non-zero mean.

If you refer back to the original full image, you can see that there was a leftover gradient in the background. This un-subtracted background, whose origin we do not yet know, is likely the cause of the excess background light. Note this sort of feature during your reduction, you'll have to explain it!

Masked SNR

Index

Finally, we return to the masked sum and SNR calculation:

In [37]:
```
print(f"Masked measurement is {masked_star_sum:0.1f} +/- {masked_star_sum_error:0
      f" for SNR={masked_star_sum/masked_star_sum_error:0.1f}")
```

```
Masked measurement is 35316.2 +/- 563.6 for SNR=62.7
```

By masking appropriately, we can increase the signal-to-noise ratio substantially.

However, be wary! If, by masking, you exclude some pixels with signal from the star, you will measure its flux inaccurately! You may end up with a *biased* flux measurement, which can lead to erroneous conclusions!

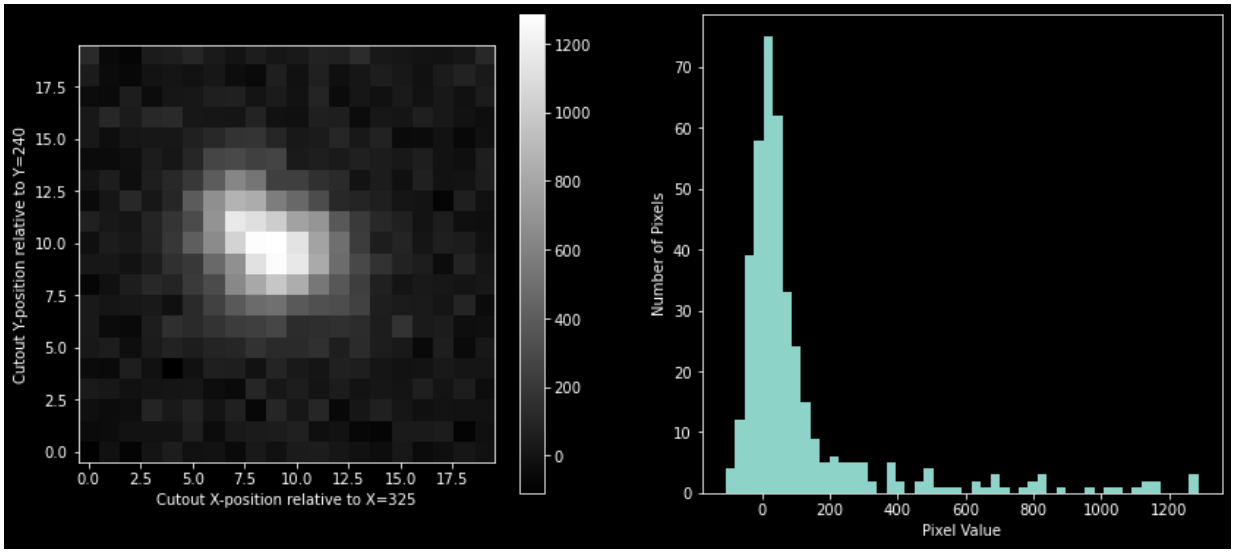
This masked measurement is still not the *optimal* measurement you can make. If you know precisely the shape of the PSF, you can perform *optimal extraction* as part of *PSF photometry*. We will not do that here, as it requires many additional steps, but we may come back later.

Photometry - not just single-star!

Index

Now, we need to repeat that exercise for some other stars. We'll be terser this time through:

In [141]:
```
cutout_star2 = sky_subtracted_pleiades_clear[240:260, 325:345, ]
pl.figure(figsize=(14,6))
im = pl.subplot(1,2,1).imshow(cutout_star2, cmap='gray', origin='lower')
pl.xlabel("Cutout X-position relative to X=325")
pl.ylabel("Cutout Y-position relative to Y=240")
pl.colorbar(im)
_=pl.subplot(1,2,2).hist(cutout_star2.ravel(), bins=50)
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



In [142]:
```
print(f"Star 2 measurement is {cutout_star2.sum():0.1f} +/- {(cutout_star2+sky_st
      f"(SNR = {cutout_star2.sum()/(cutout_star2+sky_stddev**2).sum()**0.5:0.1f}
```

```
Star 2 measurement is 45902.4 +/- 1084.2(SNR = 42.3)
```

In [143]:
```
mask2 = cutout_star2 > 2*sky_stddev
masked_star_sum2 = cutout_star2[mask2].sum()
masked_star_sum_error2 = (cutout_star2[mask2].sum() + sky_stddev**2*mask2.sum())*
print(f"Masked measurement for star 2 is {masked_star_sum2:0.1f} +/- {masked_star
      f" for SNR={masked_star_sum2/masked_star_sum_error2:0.1f}")
```

```
Masked measurement for star 2 is 40898.4 +/- 563.6 for SNR=72.6
```

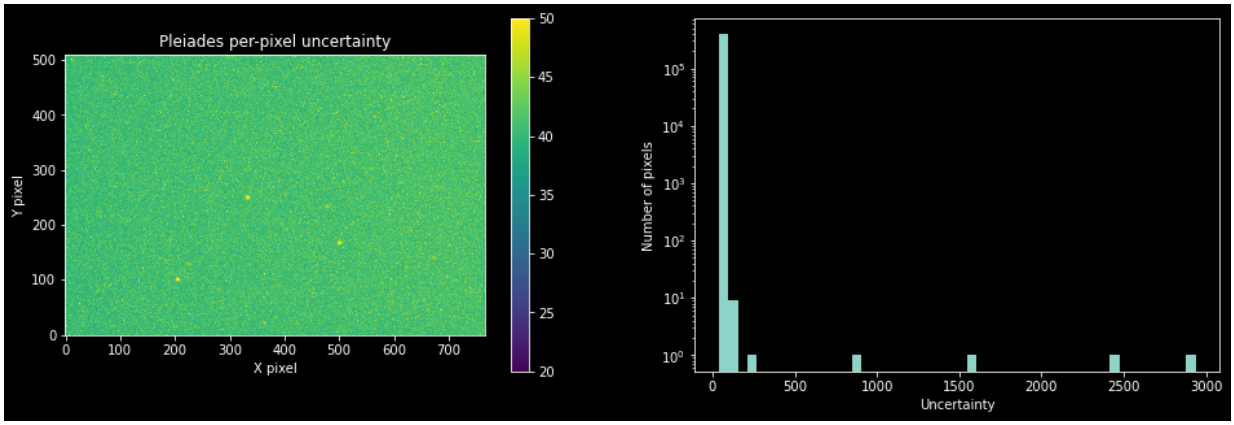
Exercise: Obtain flux measurements for two more stars

Index

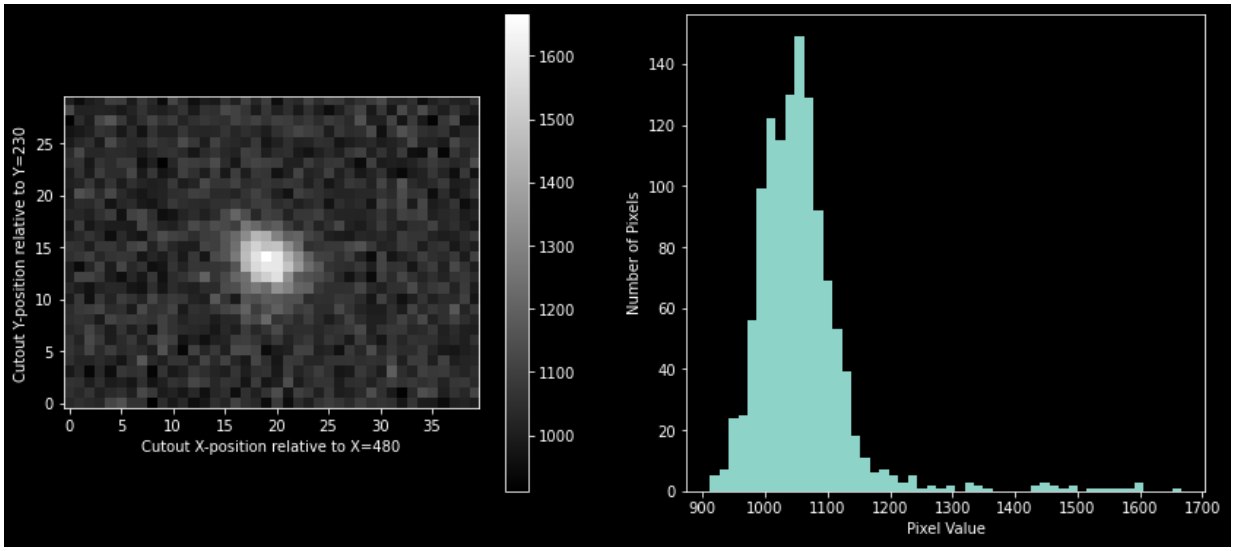
Follow the process described above to obtain flux measurements for two additional stars. Identify them yourself from the image and print out the measurement and the signal-to-noise ratio as we did above.

Compare the SNR you obtain for your star to the SNR we obtained above. Is it what you expect? (Hint: we expect a higher signal-to-noise ratio for a brighter star in the same image)

In [144]:
```
pl.figure(figsize=(16,5))
im = pl.subplot(1,2,1).imshow(pleiades_clear_darksub_flattened_uncertainty,
                              origin='lower', interpolation='none', vmax=50, vmin
pl.colorbar(im)
_=pl.title("Pleiades per-pixel uncertainty")
_=pl.xlabel("X pixel")
_=pl.ylabel("Y pixel")

_ = pl.subplot(1,2,2).hist(pleiades_clear_darksub_flattened_uncertainty.ravel(),
_ = pl.xlabel("Uncertainty")
_ = pl.ylabel("Number of pixels")
```



In [166]:
```
cutout_star3 = pleiades_clear_darksub_flattened[220:250, 460:500]
pl.figure(figsize=(14,6))
im = pl.subplot(1,2,1).imshow(cutout_star3, cmap='gray', origin='lower')
pl.xlabel("Cutout X-position relative to X=480")
pl.ylabel("Cutout Y-position relative to Y=230")
pl.colorbar(im)
_=pl.subplot(1,2,2).hist(cutout_star3.ravel(), bins=50)
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



In [167]:
```
star_sum3 = cutout_star3.sum()
print("Star sum:", star_sum3)
print("Size:", cutout_star3.size)

star_sum_error3 = (cutout_star3.size * sky_stddev**2)**0.5
print("Star sum error:", star_sum_error3)
```

```
Star sum: 1271062.6995635987
Size: 1200
Star sum error: 1840.7894915192526
```

In [168]:
```
star_sum3 / star_sum_error3
```

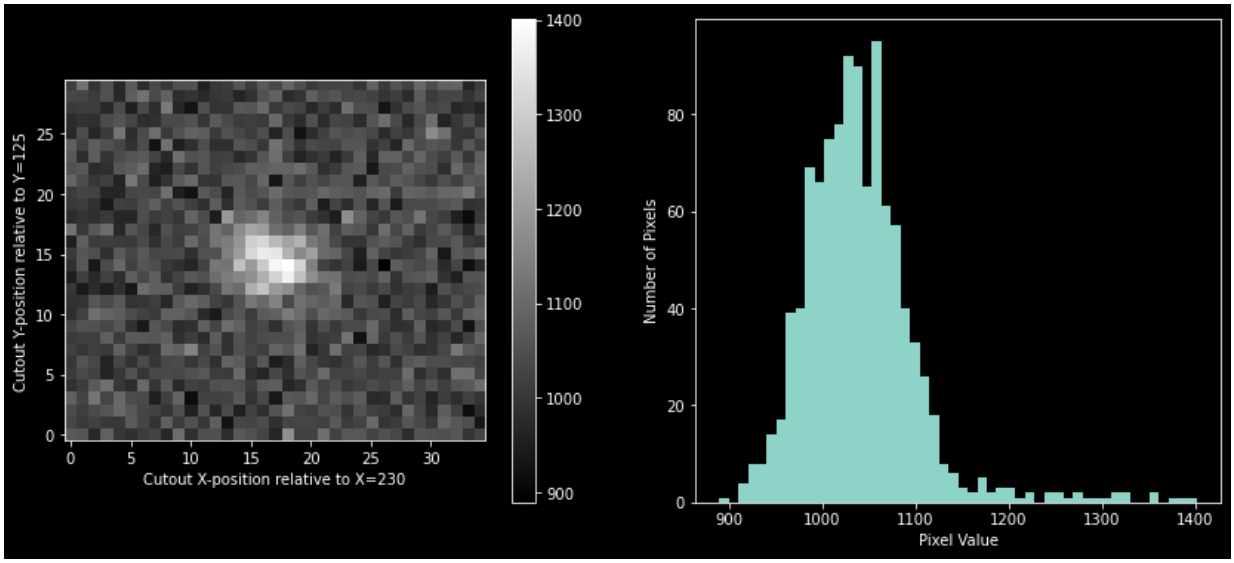
Out[168]:
```
690.4986721292921
```

In [169]:
```
per_pixel_electron_counts_star_cutout3 = cutout_star3
```

In [170]:
```
print(f"Star 2 measurement is {cutout_star3.sum():0.1f} +/- {(cutout_star3+sky_st
      f"(SNR = {cutout_star3.sum()/(cutout_star3+sky_stddev**2).sum()**0.5:0.1f})
```

```
Star 2 measurement is 1271062.7 +/- 2158.6(SNR = 588.8)
```

In [156]:
```
cutout_star4 = pleiades_clear_darksub_flattened[115:145, 210:245]
pl.figure(figsize=(14,6))
im = pl.subplot(1,2,1).imshow(cutout_star4, cmap='gray', origin='lower')
pl.xlabel("Cutout X-position relative to X=230")
pl.ylabel("Cutout Y-position relative to Y=125")
pl.colorbar(im)
_=pl.subplot(1,2,2).hist(cutout_star4.ravel(), bins=50)
_=pl.xlabel("Pixel Value")
_=pl.ylabel("Number of Pixels")
```



In [157]:
```
star_sum4 = cutout_star4.sum()

print("Star sum:", star_sum4)
print("Size:", cutout_star4.size)

star_sum_error4 = (cutout_star4.size * sky_stddev**2)**0.5
print("Star sum error:", star_sum_error4)
```

```
Star sum: 1092164.1557052564
Size: 1050
Star sum error: 1721.9008996097148
```

In [158]:
```
star_sum4 / star_sum_error4
```

Out[158]:
```
634.2781724272319
```

In [159]:
```
per_pixel_electron_counts_star_cutout4 = cutout_star4
```

In [160]:
```
print(f"Star 2 measurement is {cutout_star4.sum():0.1f} +/- {(cutout_star4+sky_st
      f"(SNR = {cutout_star4.sum()/(cutout_star4+sky_stddev**2).sum()**0.5:0.1f})
```

```
Star 2 measurement is 1092164.2 +/- 2014.2(SNR = 542.2)
```

Programming more efficiently

Index

An earlier version of this notebook included some typos in one of the cells above. I had `cutout_star2[mask]` instead of `cutout_star2[mask2]`.

This is the sort of programming error that occurs frequently when you copy and paste code. Generally, if you're copying and pasting, there's a better way!

One of the most important programming concepts is `functions`. These are, very coarsely and broadly, ways to effectively copy & paste code without copying & pasting.

If you have any operation that you want to repeat exactly, or repeat changing only a small thing (say, a coordinate...), you can write a function to do that.

In this case, we want a function to measure the star's flux and the error on the star's flux. We can do this by... well, copying and pasting what we did at first, but now generalizing.

The function will take some inputs and give some outputs. It helps to start by defining what those are (though we can modify that as we go). We know we want the two measured values, so we can start with those:

```
def calculate_star_flux_and_error(inputs):
    return masked_star_sum, masked_star_sum_error
```

Once we know what we want, we can work backwards to see what we need to do to get it. We need to have the *cutout* and the *mask*:

```
def calculate_star_flux_and_error(inputs):
    mask = cutout_star > 2*sky_stddev
    masked_star_sum = cutout_star[mask].sum()
    masked_star_sum_error = (cutout_star[mask].sum() + sky_stddev**2*mas
k.sum())**0.5

    return masked_star_sum, masked_star_sum_error
```

So what variables does the function need? The undefined ones: at least the `sky_stddev` and the `cutout_star`. However, both of these are derived from the parent image: really, we only need the parent image, then we can calculate both.

This is how we calculated the cutout and the stddev:

```
cutout_star = pleiades_clear_darksub_flattened[158:178, 492:512]

sky_stddev = pleiades_clear_darksub_flattened.std()
```

We can give those as inputs to the function, and now we have a working function:

In [161]:
```
def calculate_star_flux_and_error(cutout_star, sky_stddev):

    mask = cutout_star > 2*sky_stddev
    masked_star_sum = cutout_star[mask].sum()
    masked_star_sum_error = (cutout_star[mask].sum() + sky_stddev**2*mask.sum())*

    return masked_star_sum, masked_star_sum_error
```

In [162]:
```
calculate_star_flux_and_error(cutout_star, sky_stddev)
```

Out[162]:
```
(1092164.1557052564, 2014.2261203206558)
```

Now that we have this function, it's pretty easy to repeat our measurement for the second star:

In [163]:
```
calculate_star_flux_and_error(cutout_star2, sky_stddev)
```

Out[163]:
```
(40898.35182370345, 563.5834790968116)
```

Exercise: Use the function to repeat the exercise on another two stars

Index

In [164]:
```
calculate_star_flux_and_error(cutout_star3, sky_stddev)
```

Out[164]:
```
(1271062.6995635987, 2158.6034030482087)
```

In [165]:
```
calculate_star_flux_and_error(cutout_star4, sky_stddev)
```

Out[165]:
```
(1092164.1557052564, 2014.2261203206558)
```