In [1]:
```
from __future__ import print_function
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
from IPython.display import HTML, YouTubeVideo
import matplotlib.patches as patches
from matplotlib.lines import Line2D
# import ganymede
# ganymede.configure('uav.beaver.works')
```

Enter your name below and run the cell:

Individual cells can be run with `Ctrl + Enter`

In [2]:
```
# ganymede.name('YOUR NAME HERE')
# def check(p):
#     ganymede.update(p,True)
# check(0)
```

https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/more-on-regression/v/squared-error-of-regression-line (https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/more-on-regression/v/squared-error-of-regression-line)

Note: All Khan Academy content is available for free at khanacademy.org

In [3]:
```
YouTubeVideo('6OvhLPS7rj4', width=560, height=315)
```

Out[3]:

In [4]:
```
YouTubeVideo('mIx2Oj5y9Q8', width=560, height=315)
```

Out[4]:

In [5]:
```
YouTubeVideo('f6OnoxctvUk', width=560, height=315)
```

Out[5]:

In [6]:
```
YouTubeVideo('u1HhUB3NP8g', width=560, height=315)
```

Out[6]:

In [7]:
```
YouTubeVideo('8RSTQl0bQuw', width=560, height=315)
```

Out[7]:

In [8]:
```
YouTubeVideo('GAmzwIkGFgE', width=560, height=315)
```

Out[8]:

The last video is optional

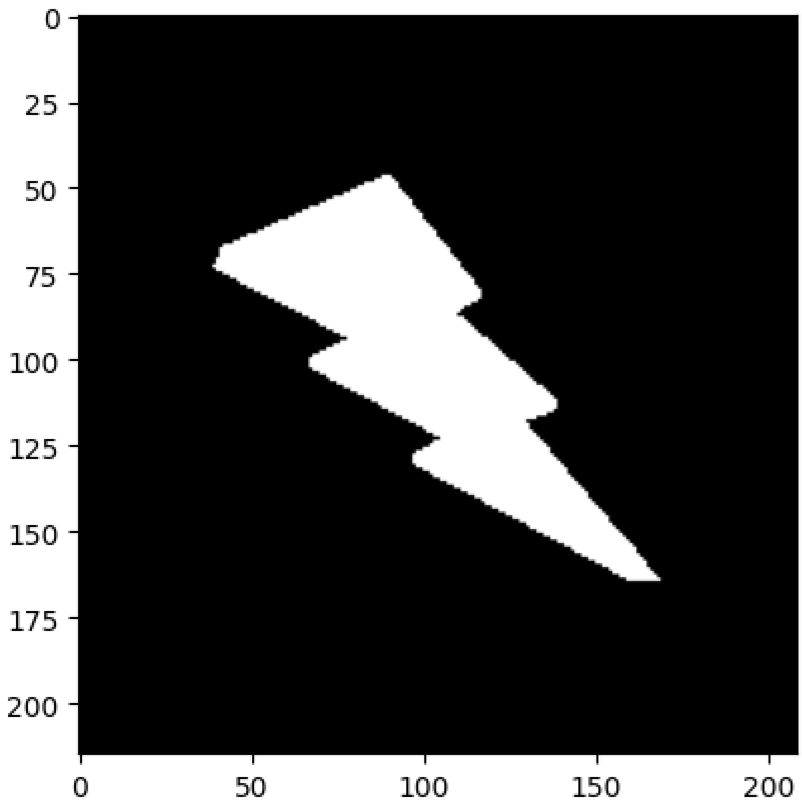
In [9]:
```
YouTubeVideo('ww_yT9ckPWw', width=560, height=315)
```

Out[9]:

In [10]:
```
lightningbolt = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSC
_, lightningbolt = cv2.threshold(lightningbolt,150,255,cv2.THRESH_BINARY
print(lightningbolt.shape)
fig,ax = plt.subplots()
ax.imshow(lightningbolt, cmap='gray')
```

```
(215, 209)
```

Out[10]:
```
<matplotlib.image.AxesImage at 0x121ecee80>
```



In [ ]:

In [11]:
```
bolt = np.argwhere(lightningbolt)
print(bolt)
```

```
[[ 47  88]
 [ 47  89]
 [ 47  90]
 ...
 [164 166]
 [164 167]
 [164 168]]
```

Linear Regression

$$m = \frac{\bar{x}\bar{y} - \overline{xy}}{(\bar{x})^2 - \overline{x^2}}$$

$$b = \bar{y} - m\bar{x}$$

Question: how can we extract the xs and ys separately from the result of argwhere?

Hint: review numpy slicing by columns and rows

In [12]:
```
# TODO
# We can iterate through all the pairs that argwhere gives us, then put 
```

Question: Why would we want to convert x and y points from int values to floats?

In [13]:
```
# TODO
# floats are easier to analyze since they are continuous (essentially)
```

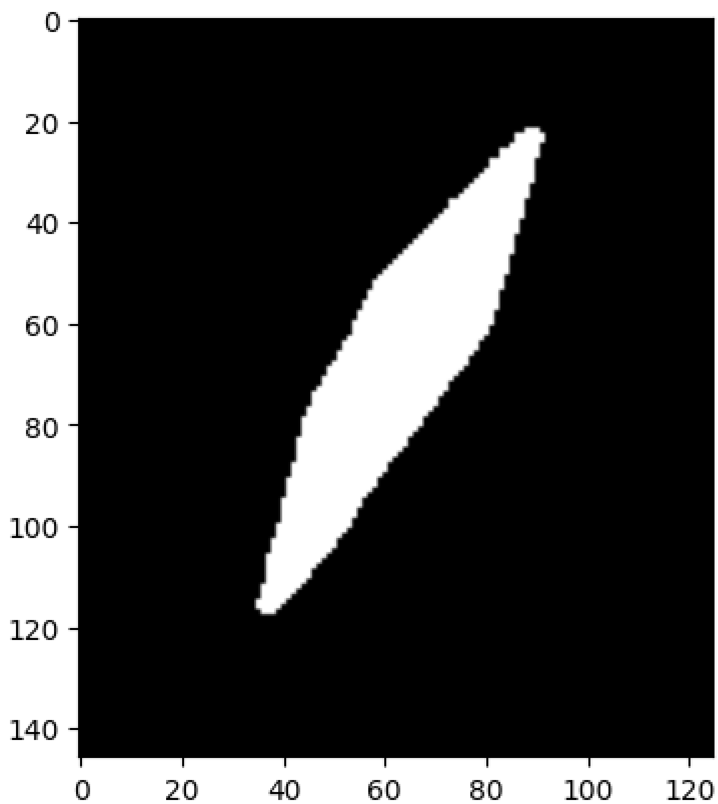
In [44]:
```
def calculate_regression(points): # input is the result of np.argwhere
    # convert points to float
    points = points.astype(float) #TODO (see astype, https://docs.scipy.
    # print(points)
    l=len(points)
    xs=[]
    ys=[]
    for i in range(l):
        xs.append(points[i][0])
        ys.append(points[i][1])
    xys=np.multiply(xs,ys)
    x_squared=np.multiply(xs,xs)
    x_mean = np.mean(xs)
    y_mean = np.mean(ys)

    xy_mean = np.mean(xys)

    x_squared_mean = np.mean(x_squared)

    m =(x_mean*y_mean-xy_mean)/(x_mean*x_mean-x_squared_mean)

    b = y_mean-m*x_mean

    return (m,b)
```

The intercept we calculated, b, may be outside of the pixel space of the image, so we must find two points inside of pixel space, (x_1, y_1) and $(x2, y2)$ which will allow us to plot our regression line on the image. It may be best to choose points on the regression line which also occur on the boundaries/extrema of the image.

In [33]:
```
def find_inliers(m, b, shape):
    x1, y1, x2, y2 = max(0,-b)/m, max(0,-b)+b, min(m*shape[0],shape[1]-b
    return [x1,y1,x2,y2]
```

In [51]:
```
blob = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
print(blob.shape)

_, blob = cv2.threshold(blob,125,255,cv2.THRESH_BINARY)
fig,ax = plt.subplots()
ax.imshow(blob, cmap='gray');
```

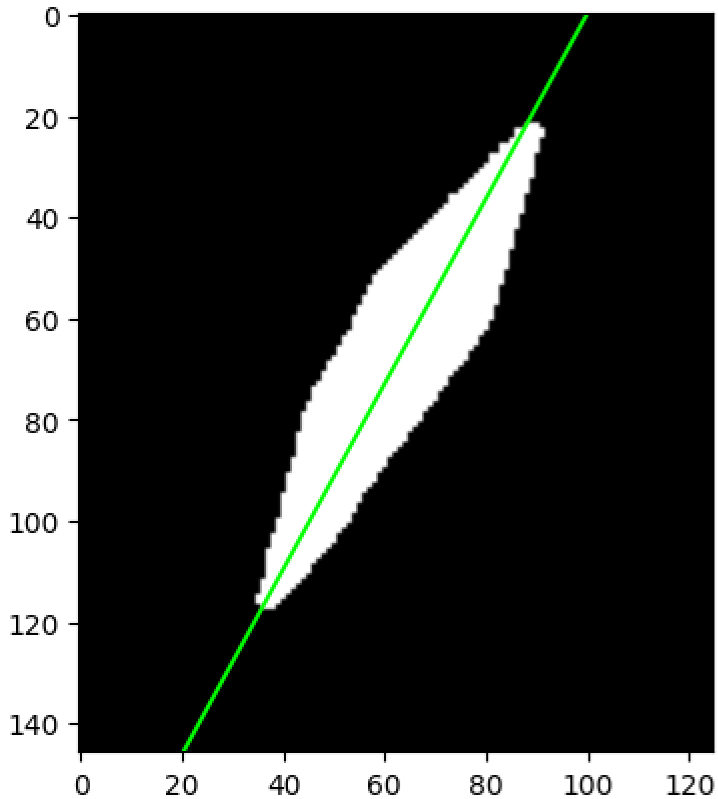
```
(146, 125)
```



In [52]:
```
m,b = calculate_regression(np.argwhere(blob))
pts = find_inliers(m,b, blob.shape)
print(blob.shape)
print(m,b)
print(pts)
```

```
(146, 125)
-0.5467322545592981 99.8488561411747
[-0.0, 99.8488561411747, 146.0, 20.025946975517186]
```

In [54]:
```
# below is an example of how to draw a random line from (10,25) to (10,55
# TODO: replace this with the result of find_inliers
# -- pay attention to the directions of the x and y axes
#    in image space, row-column space, and cartesian space
# Look at the help function for Line2D below

fig,ax = plt.subplots()
ax.imshow(blob, cmap='gray');
regression = Line2D([pts[1],pts[3]],[pts[0],pts[2]], color='lime')
ax.add_line(regression);
```



In [41]:
```
Line2D?
```

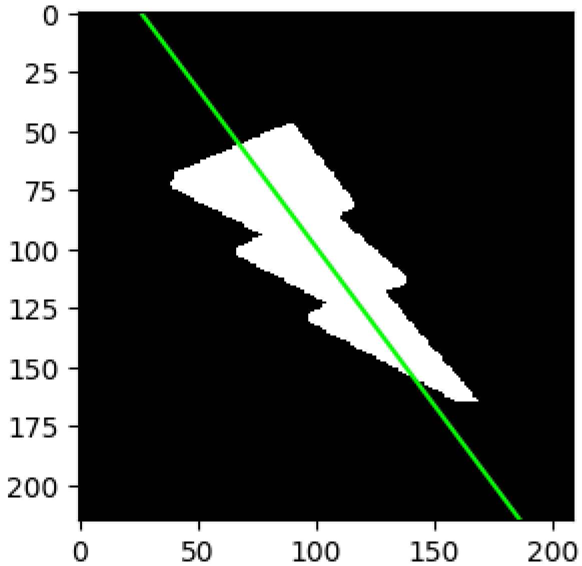
```
Init signature:
Line2D(
    xdata,
    ydata,
    *,
    linewidth=None,
    linestyle=None,
    color=None,
    gapcolor=None,
    marker=None,
    markersize=None,
    markeredgewidth=None,
    markeredgecolor=None,
    markerfacecolor=None,
    markerfacecoloralt='none',
    fillstyle=None,
    antialiased=None,
    dash_capstyle=None,
    solid_capstyle=None,
    dash_joinstyle=None,
    solid_joinstyle=None,
    pickradius=5,
    drawstyle=None,
    markevery=None,
    **kwargs,
)
Docstring:
A line - the line can have both a solid linestyle connecting all
the vertices, and a marker at each vertex.  Additionally, the
drawing of the solid line is influenced by the drawstyle, e.g., one
can create "stepped" lines in various styles.
Init docstring:
Create a `.Line2D` instance with *x* and *y* data in sequences of
*xdata*, *ydata*.

Additional keyword arguments are `.Line2D` properties:

Properties:
    agg_filter: a filter function, which takes a (m, n, 3) float array
and a dpi value, and returns a (m, n, 3) array and two offsets from the
bottom left corner of the image
    alpha: scalar or None
    animated: bool
    antialiased or aa: bool
    clip_box: `~matplotlib.transforms.BboxBase` or None
    clip_on: bool
    clip_path: Patch or (Path, Transform) or None
    color or c: :mpltype:`color`
    dash_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
    dash_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
    dashes: sequence of floats (on/off ink in points) or (None, None)
    data: (2, N) array or two 1D arrays
    drawstyle or ds: {'default', 'steps', 'steps-pre', 'steps-mid', 'st
eps-post'}, default: 'default'
    figure: `~matplotlib.figure.Figure`
    fillstyle: {'full', 'left', 'right', 'bottom', 'top', 'none'}
    gapcolor: :mpltype:`color` or None
```

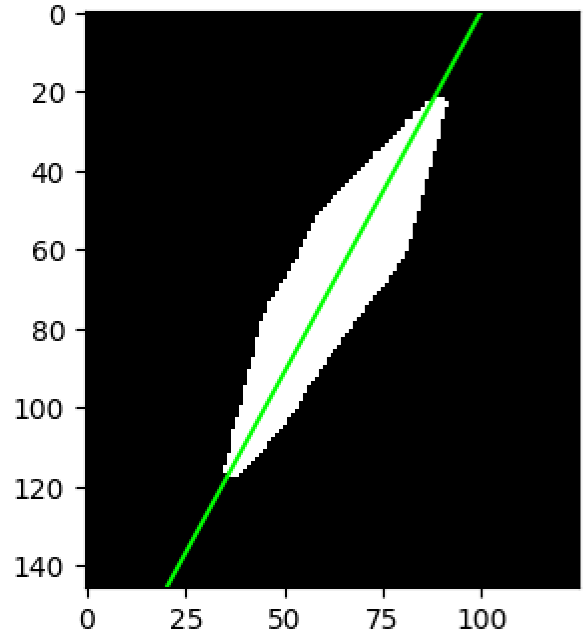
```
    gid: str
    in_layout: bool
    label: object
    linestyle or ls: {'-', '--', '-.', ':', '', (offset, on-off-seq),
...}
    linewidth or lw: float
    marker: marker style string, `~.path.Path` or `~.markers.MarkerStyl
e`
    markeredgecolor or mec: :mpltype:`color`
    markeredgewidth or mew: float
    markerfacecolor or mfc: :mpltype:`color`
    markerfacecoloralt or mfcalt: :mpltype:`color`
    markersize or ms: float
    markevery: None or int or (int, int) or slice or list[int] or float
or (float, float) or list[bool]
    mouseover: bool
    path_effects: list of `.AbstractPathEffect`
    picker: float or callable[[Artist, Event], tuple[bool, dict]]
    pickradius: float
    rasterized: bool
    sketch_params: (scale: float, length: float, randomness: float)
    snap: bool or None
    solid_capstyle: `.CapStyle` or {'butt', 'projecting', 'round'}
    solid_joinstyle: `.JoinStyle` or {'miter', 'round', 'bevel'}
    transform: unknown
    url: str
    visible: bool
    xdata: 1D array
    ydata: 1D array
    zorder: float

See :meth:`set_linestyle` for a description of the line styles,
:meth:`set_marker` for a description of the markers, and
:meth:`set_drawstyle` for a description of the draw styles.
File:           ~/Library/Python/3.9/lib/python/site-packages/matplotli
b/lines.py
Type:           type
Subclasses:     AxLine, Line3D
```

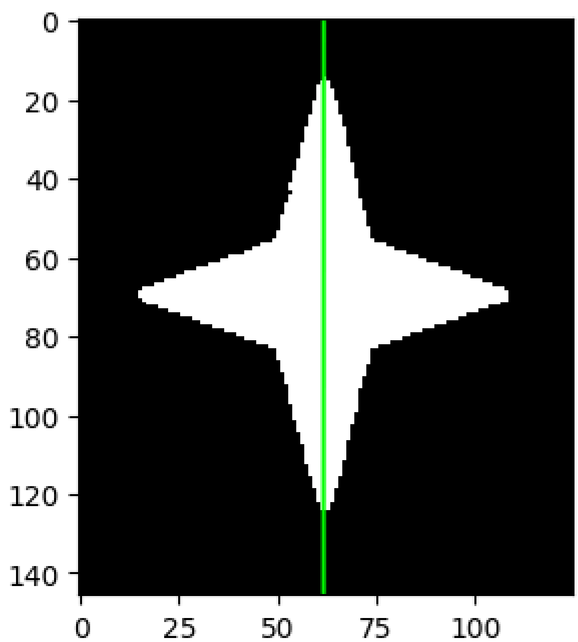
TODO

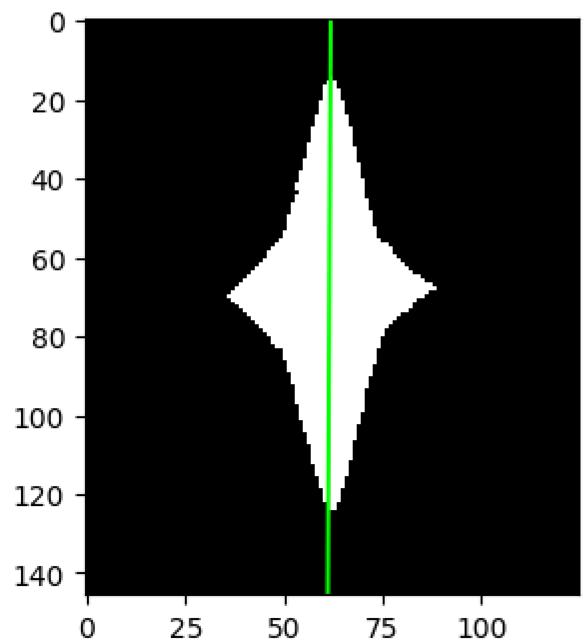
1. Run your linear regression algorithm on the following images.
2. Plot each of the results.
3. Include each result in your submitted PDF.

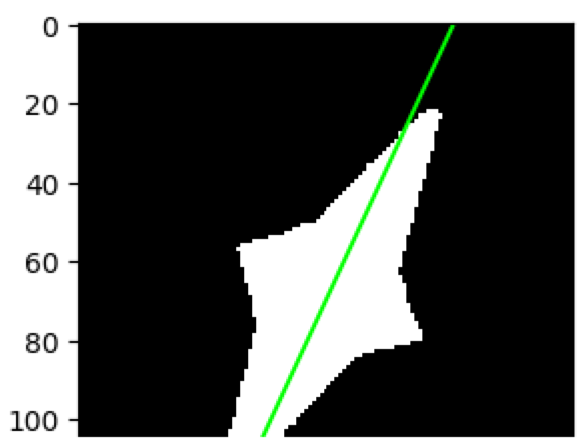
In [56]:
```
lightningbolt      = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GI
blob               = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
star               = cv2.imread('shapes/star.png', cv2.IMREAD_GRAYSCALE)
squishedstar       = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GR/
squishedturnedstar = cv2.imread('shapes/squishedturnedstar.png', cv2.IMRI
letterj            = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCAI

images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, l(
fig,ax = plt.subplots(nrows=3, ncols=2)
for a,i in zip(ax.flatten(), images):
    _, i = cv2.threshold(i,125,255,cv2.THRESH_BINARY)
    a.imshow(i, cmap='gray', interpolation='none')
    m,b = calculate_regression(np.argwhere(i))
    pts = find_inliers(m,b, i.shape)
    # print(i.shape)
    # print(m,b)
    # print(pts)
    regression = Line2D([pts[1],pts[3]],[pts[0],pts[2]], color='lime')
    a.add_line(regression)
fig.set_size_inches(7,14)
```

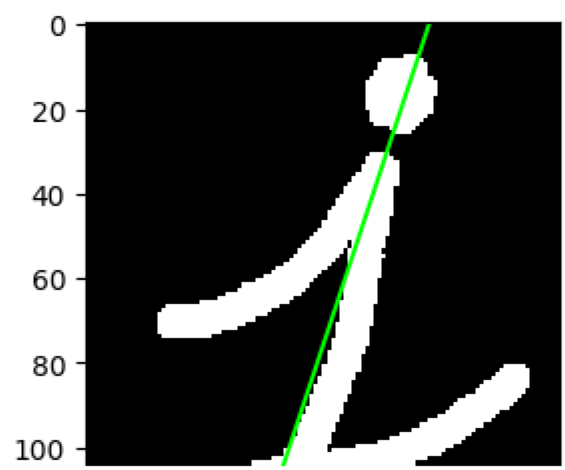

0
25
50
75
100
125
150
175
200
0 50 100 150 200

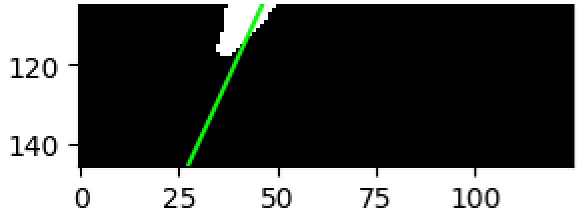

0
20
40
60
80
100
120
140
0 25 50 75 100

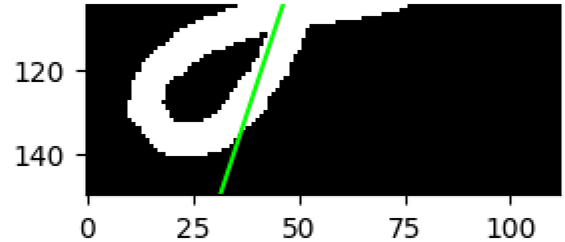

0
20
40
60
80
100
120
140
0 25 50 75 100


0
20
40
60
80
100
120
140
0 25 50 75 100


0
20
40
60
80
100


0
20
40
60
80
100





When you are done:

You should have six images with regression lines plotted on top of them.

1. Double-check that you filled in your name at the top of the notebook!
2. Click `File -> Export Notebook As -> PDF`
3. Email the PDF to `YOURTEAMNAME@beaver.works`

Stretch goal

Implement a machine learning algorithm!

Random **Sa**mple **C**onsensus, commonly referred to as *RANSAC*, is one of the most widely used machine learning algorithms. In essence, it is a 'guess and check' algorithm. Take a small random sample of your data - two points in this case. Next, define a line through those two points. After doing so, count the number of *inliers*, or points closest to that line (euclidean distance is one way to do this).

[https://en.wikipedia.org/wiki/Random_sample_consensus (https://en.wikipedia.org/wiki/Random_sample_consensus)](https://en.wikipedia.org/wiki/Random_sample_consensus)

Implement RANSAC for linear regression, and run it on all of your images.