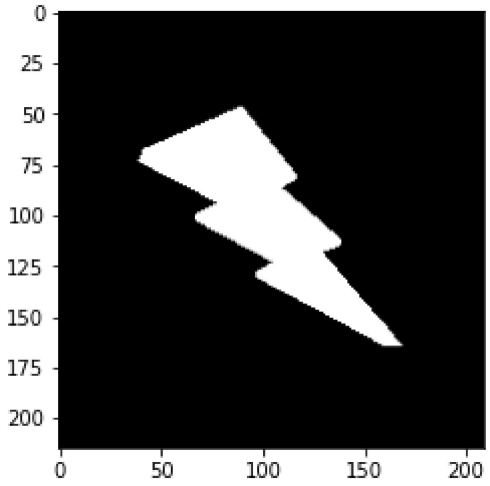
In [ ]:
```
from __future__ import print_function
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
import cv2
from IPython.display import HTML, YouTubeVideo
import matplotlib.patches as patches
from matplotlib.lines import Line2D
```

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

```
(215, 209)
```



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

Out[ ]:
```
array([[ 47,  88],
       [ 47,  89],
       [ 47,  90],
       ...,
       [164, 166],
       [164, 167],
       [164, 168]], dtype=int64)
```

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 [ ]:
```
xs = bolt[:,0]
ys = bolt[:,1]
```

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

Calculating a regression line will involve operations with decimals, and int values to not have decimal points. Converting the x and y points to floats allows for greater precision because they include decimals.

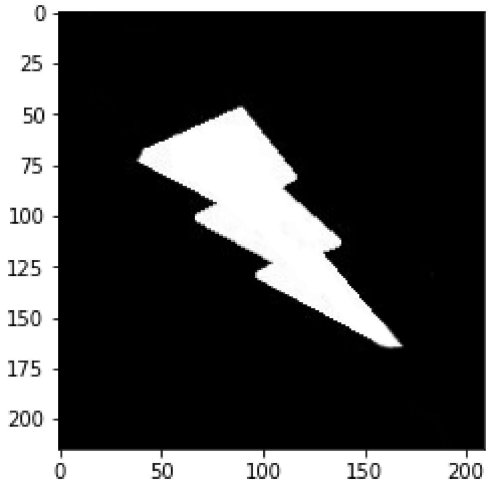
In [ ]:
```
def calculate_regression(points): # input is the result of np.argwhere
    # convert points to float
    points = points.astype(np.float)

    xs = points[:,0]
    ys = points[:,1]
    x_mean = np.mean(xs)
    y_mean = np.mean(ys)

    xy_mean = np.mean(np.multiply(xs,ys))

    x_squared_mean = np.mean(np.power(xs,2))
    m = ((x_mean * y_mean) - xy_mean)/(x_mean**2 - 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 [ ]:
```
def find_inliers(m, b, shape):
    x1, y1, x2, y2 = 0, b, shape[1], shape[1]*m + b
    return((x1,y1,x2,y2))
```

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

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

```
(215, 209)
```

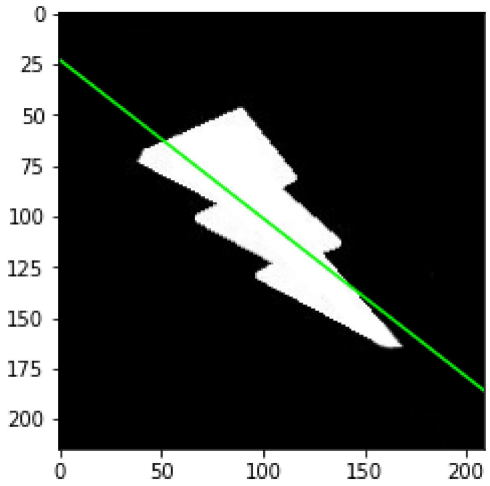


In [ ]:
```
m,b = calculate_regression(np.argwhere(lightningbolt))
print(m,b)
inliers = find_inliers(m,b, lightningbolt.shape)
print(inliers)
```

```
0.780694883465611 22.81831139826832
(0, 22.81831139826832, 209, 185.983542042581)
```

```
C:\Users\owent\AppData\Local\Temp\ipykernel_24136\2596586151.py:3: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning,
use `float` by itself. Doing this will not modify any behavior and is safe. If you sp
ecifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
lease/1.20.0-notes.html#deprecations
  points = points.astype(np.float)
```

In [ ]:
```
# 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(lightningbolt, cmap='gray');
regression = Line2D([inliers[0],inliers[2]],[inliers[1],inliers[3]], color='lime')
ax.add_line(regression);
```

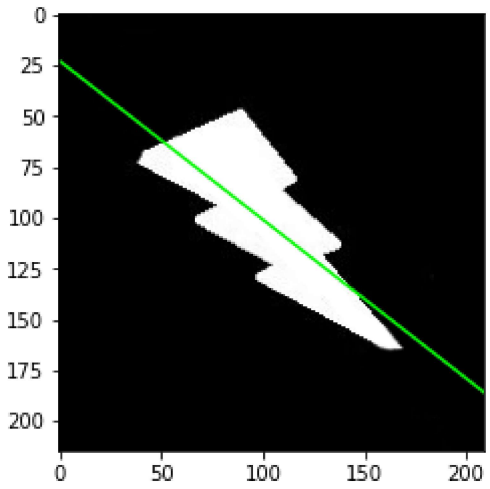


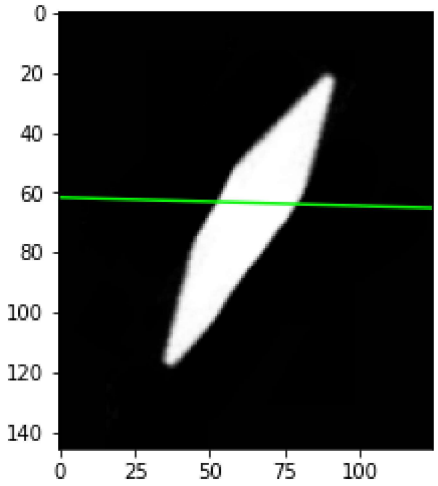
TODO

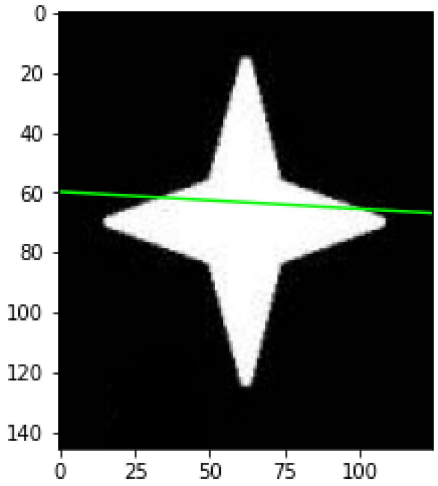
1. Run your linear regression algorithm on the following images.
2. Plot each of the results.
3. Include each result in your submitted PDF.

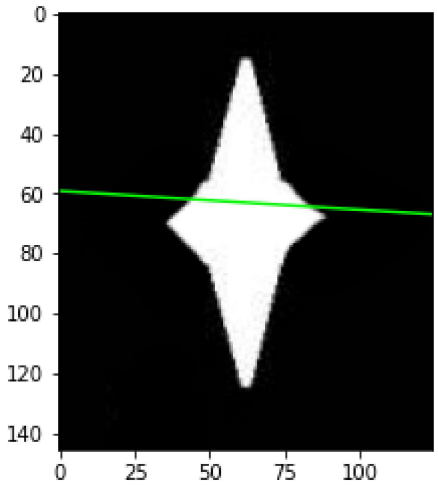
In [ ]:
```
lightningbolt      = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GRAYSCALE)
blob               = cv2.imread('shapes/blob.png', cv2.IMREAD_GRAYSCALE)
star               = cv2.imread('shapes/star.png', cv2.IMREAD_GRAYSCALE)
squishedstar       = cv2.imread('shapes/squishedstar.png', cv2.IMREAD_GRAYSCALE)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

images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, letterj]

for img in images:
    m,b = calculate_regression(np.argwhere(img))
    inliers = find_inliers(m,b, img.shape)
    fig,ax = plt.subplots()
    ax.imshow(img, cmap='gray');
    regression = Line2D([inliers[0],inliers[2]],[inliers[1],inliers[3]], color='lime')
    ax.add_line(regression);
```

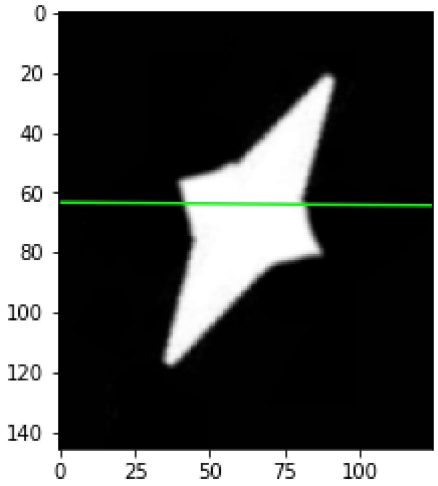
```
C:\Users\owent\AppData\Local\Temp\ipykernel_24136\2596586151.py:3: DeprecationWarnin
g: `np.float` is a deprecated alias for the builtin `float`. To silence this warning,
use `float` by itself. Doing this will not modify any behavior and is safe. If you sp
ecifically wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/re
lease/1.20.0-notes.html#deprecations
  points = points.astype(np.float)
```

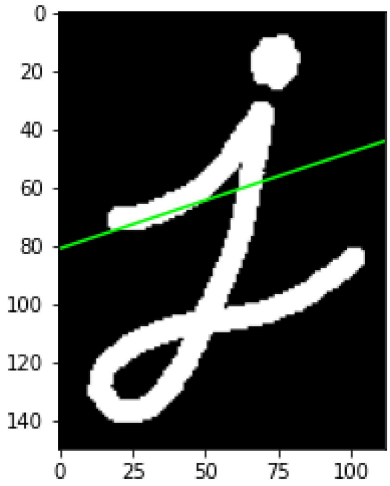



0
20
40
60
80
100
120
140
0
25
50
75
100


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





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

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