In [68]:
```
%matplotlib inline
from __future__ import print_function
#import ganymede
#ganymede.configure('uav.beaver.works')
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
```

In [69]:
```
def check(p): pass
check(0)
```

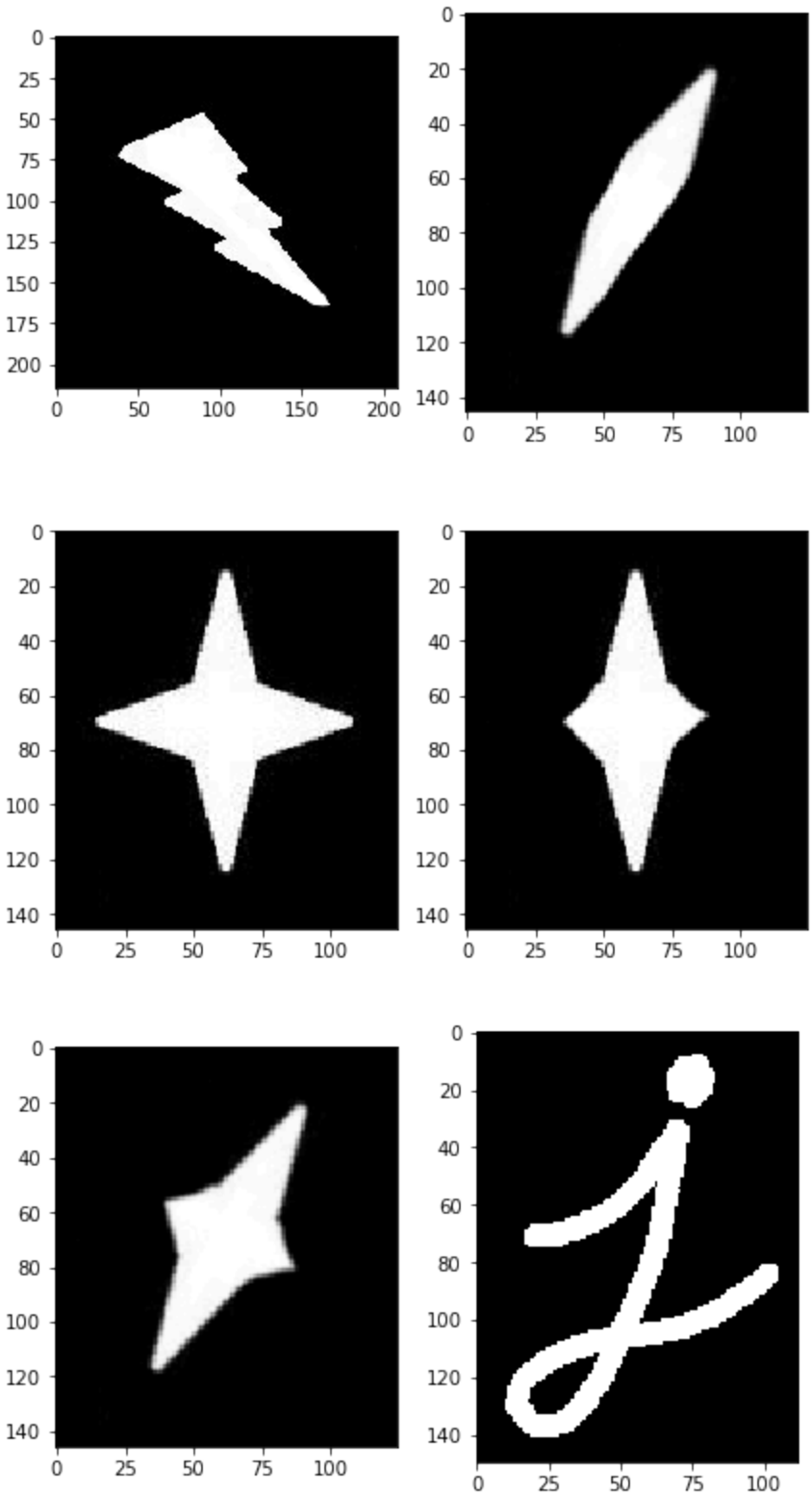
Note

`cv2.imshow()` will not work in a notebook, even though the OpenCV tutorials use it. Instead, use `plt.imshow` and family to visualize your results.

In [70]:
```
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_GRAYSCA
letterj            = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCALE)

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

fig,ax = plt.subplots(nrows=3, ncols=2)
for a,i in zip(ax.flatten(), images):
    a.imshow(i, cmap='gray', interpolation='none');
fig.set_size_inches(7,14);
```



In [71]:
```
intensity_values = set(lightningbolt.flatten())
print(len(intensity_values))
```

75

Question:

What would you expect the value to be, visually? What explains the actual value?

In [5]:
```
# Visually, I would expect the value to be around 3-4, as although the image is not
```

Thresholding

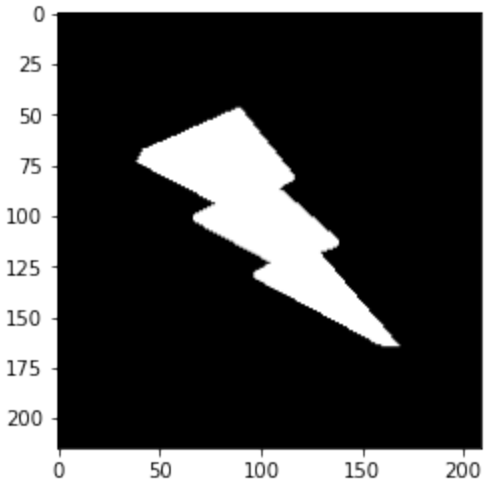
https://docs.opencv.org/3.4.1/d7/d4d/tutorial_py_thresholding.html

In [72]:
```
_, lightningbolt = cv2.threshold(lightningbolt,180,255,cv2.THRESH_BINARY)

intensity_values = set(lightningbolt.flatten())
print(len(intensity_values))

plt.imshow(lightningbolt, cmap='gray');
```

```
2
```



Question

What happens when the above values are used for thresholding? What is a "good" value for thresholding the above images? Why?

In [73]:
```
# When the above values are used for thresholding, most of the fully white or black
```

Exercises

Steps

1. Read each tutorial
    - Skim all parts of each tutorial to understand what each operation does
    - Focus on the part you will need for the requested transformation
2. Apply the transformation and visualize it

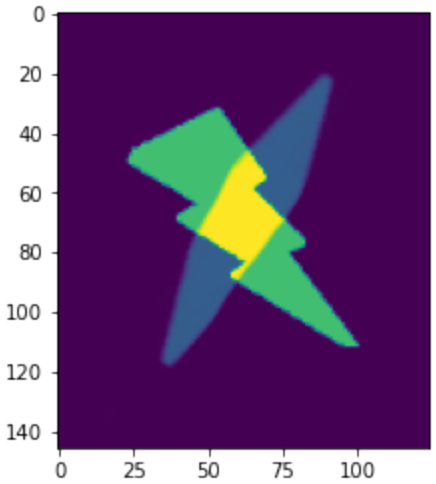
1. Blend lightningbolt and blob together

https://docs.opencv.org/3.4.1/d0/d86/tutorial_py_image_arithmetics.html

Remember: Don't use `imshow` from OpenCV, use `imshow` from `matplotlib`

In [87]:
```
resizedBolt = cv2.resize(lightningbolt, (blob.shape[1], blob.shape[0])) # blob.shap
dst = cv2.addWeighted(resizedBolt, 0.7, blob, 0.3, 0)
plt.imshow(dst)
```

Out[87]:
```
<matplotlib.image.AxesImage at 0x7ce9e71f66e0>
```

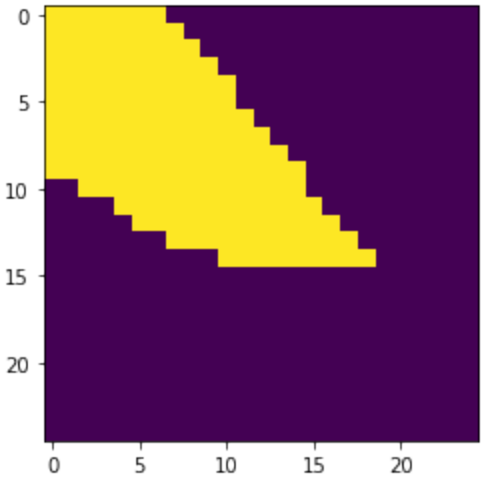


2. Find a ROI which contains the point of the lightning bolt

https://docs.opencv.org/3.4.1/d3/df2/tutorial_py_basic_ops.html

In [90]:
```
boltTip = lightningbolt[150:175, 150:175]
plt.imshow(boltTip)
```

Out[90]:
```
<matplotlib.image.AxesImage at 0x7ce9e6f9d870>
```

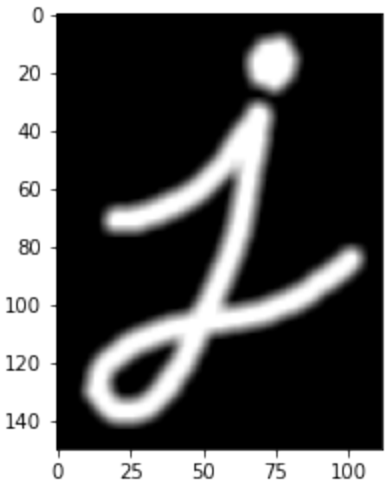


3. Use an averaging kernel on the letter j

https://docs.opencv.org/3.4.1/d4/d13/tutorial_py_filtering.html

In [93]:
```
kernelAvg = np.ones((5, 5), np.float32)/25
smoothed = cv2.filter2D(letterj, -1, kernel)
plt.imshow(smoothed, cmap='gray') # idk why I made this one gray - I guess everyone
```

Out[93]: <matplotlib.image.AxesImage at 0x7ce9e768b0a0>



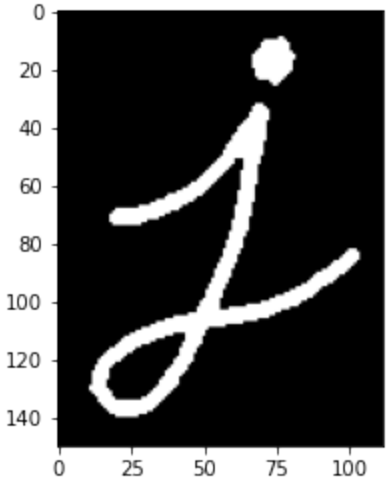
Morphology

https://docs.opencv.org/3.4.1/d9/d61/tutorial_py_morphological_ops.html

4. Perform erosion on j with a 3x3 kernel

In [101…
```
kernel3 = np.ones((3, 3), np.uint8)
erosion3 = cv2.erode(letterj, kernel3, iterations=1)
plt.imshow(erosion3, cmap='gray')
```

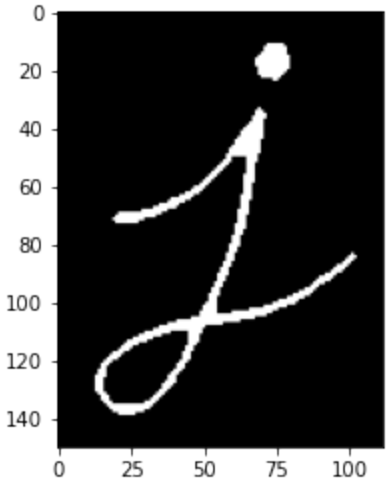
Out[101… <matplotlib.image.AxesImage at 0x7ce9e6c26d10>



5. Perform erosion on j with a 5x5 kernel

In [102…
```
kernel5 = np.ones((5, 5), np.uint8)
erosion5 = cv2.erode(letterj, kernel5, iterations=1)
plt.imshow(erosion5, cmap='gray')
```

Out[102… <matplotlib.image.AxesImage at 0x7ce9e6c91f30>



6. Perform erosion on j with **two** iterations, using a kernel size of your choice

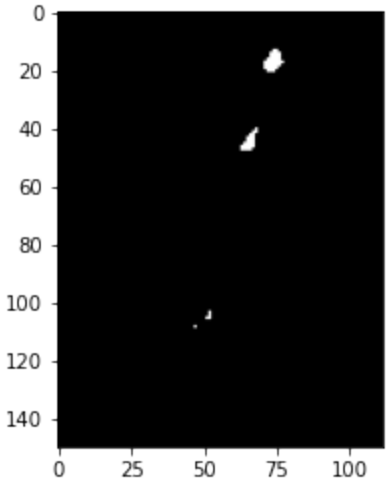
Hint: look at the OpenCV API documentation. It is possible to perform two iterations of erosion in one line of Python!

https://docs.opencv.org/3.4.1/d4/d86/group__imgproc__filter.html#gaeb1e0c1033e3f6b891a25d

In [103…
```
erosionTwice = cv2.erode(letterj, kernel5, iterations=2)
```

```
plt.imshow(erosionTwice, cmap='gray')
```

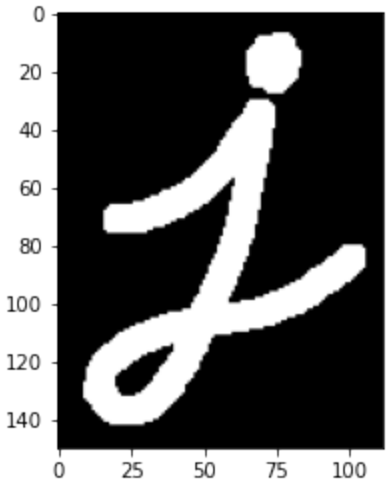
Out[103… `<matplotlib.image.AxesImage at 0x7ce9e6b04070>`



7. Perform dilation on j with a 3x3 kernel

In [105…
```
kernelDilate3 = np.ones((3, 3), np.uint8)
dilation3 = cv2.dilate(letterj, kernelDilate3, iterations=1)
plt.imshow(dilation3, cmap='gray')
```

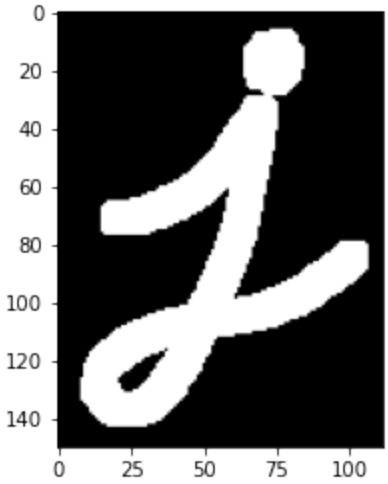
Out[105… `<matplotlib.image.AxesImage at 0x7ce9e6b744c0>`



8. Perform dilation on j with a 5x5 kernel

In [106…
```
kernelDilate5 = np.ones((5, 5), np.uint8)
dilation5 = cv2.dilate(letterj, kernelDilate5, iterations=1)
plt.imshow(dilation5, cmap='gray')
```

Out[106… `<matplotlib.image.AxesImage at 0x7ce9e6bb2920>`



9. What is the effect of kernel size on morphology operations?

In [ ]:
```
# The kernel size determines how much of the image is eroded/dilated. If the kernel
```

10. What is the difference betweeen repeated iterations of a morphology operation with a small kernel, versus a single iteration with a large kernel?

In [ ]:
```
# There doesn't seem to be a visual difference between using repeated iterations ve
```

11. Rotate the lightningbolt and star by 90 degrees

https://docs.opencv.org/3.4.1/da/d6e/tutorial_py_geometric_transformations.html

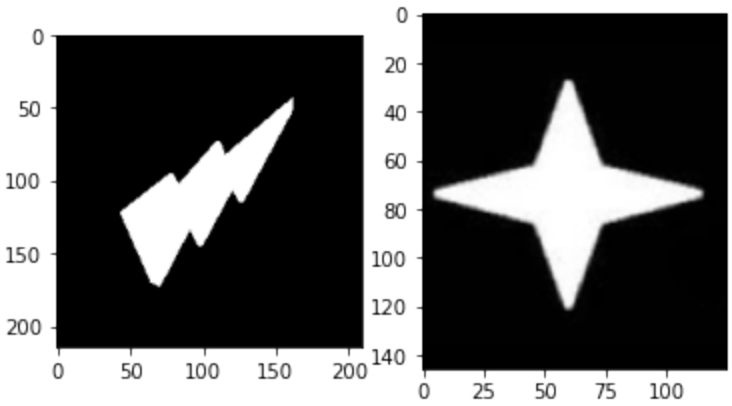
In [111…
```
rows, cols = lightningbolt.shape[:2]
M = cv2.getRotationMatrix2D((cols/2, rows/2), 90, 1)
rotatedBolt = cv2.warpAffine(lightningbolt, M, (cols, rows))

rows1, cols1 = star.shape[:2]
M1 = cv2.getRotationMatrix2D((cols1/2, rows1/2), 90, 1)
rotatedStar = cv2.warpAffine(star, M1, (cols1, rows1))

fig,ax = plt.subplots(nrows=1, ncols=2)

ax[0].imshow(rotatedBolt, cmap='gray')
ax[1].imshow(rotatedStar, cmap='gray')
```

Out[111…
```
<matplotlib.image.AxesImage at 0x7ce9e6927070>
```



12. STRETCH GOAL:

Visualize the result of Laplacian, Sobel X, and Sobel Y on all of the images. Also, produce a combined image of both Sobel X and Sobel Y for each image. Is Exercise 1 the best way to do this? Are there other options?

You should have 4 outputs (Laplacian, SobelX, SobelY, and the combination) for each input image visualized at the end.

https://docs.opencv.org/3.4.1/d5/d0f/tutorial_py_gradients.html

When you are done:

You should have one or more images for each exercise.

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`