In [9]:
```python
%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 [3]:
```python
def check(p): pass
check(0)
```

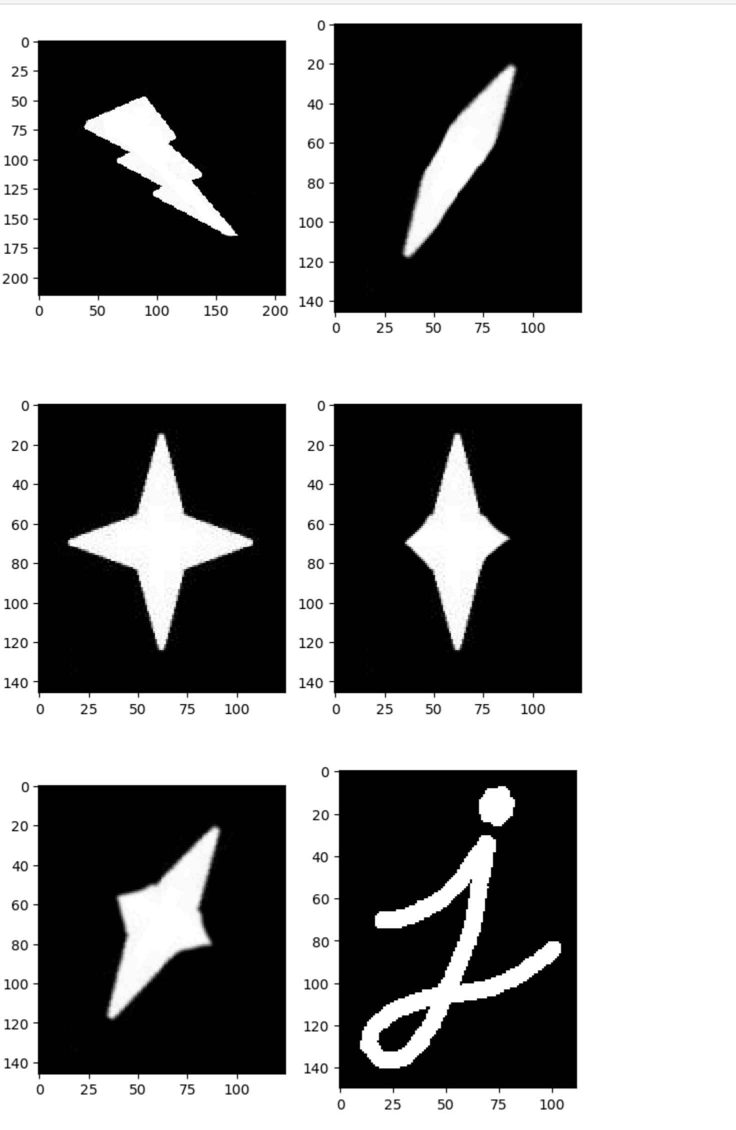
Note

`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 [36]:
```python
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]

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 [5]:
```python
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 [ ]:
```python
# TODO
# One would expect the value to higher becaues the images has more than 75 pixels, but the set() function only counts unique intensity values, of wh
```

Thresholding

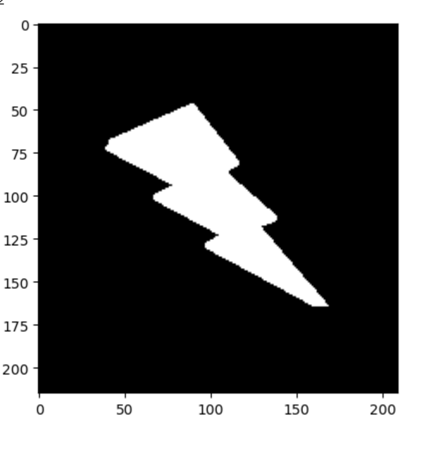
https://docs.opencv.org/3.4.1/d7/d4d/tutorial_py_thresholding.html

In [11]:
```python
_, lightningbolt = cv2.threshold(lightningbolt,127,255,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 [13]:
```python
## TODO
## They are all turned to white. 127 would be a "good" thresholding value because it is the middle of greyscale values (0-255); lighter greys will b
```

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

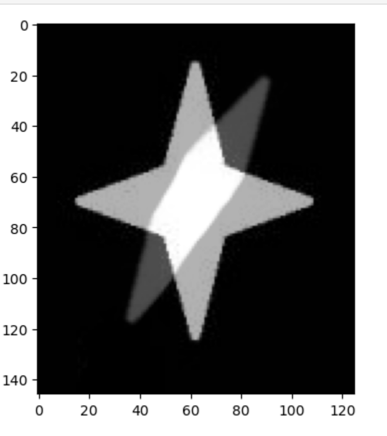
https://docs.opencv.org/3.4.1/d0/d86/tutorial_py_image_arithmetics.html

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

In [17]:
```python
# 1. Blend

dst = cv2.addWeighted(star,0.7,blob,0.3,0)

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

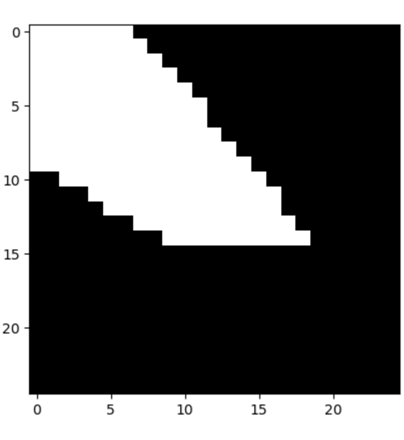


2. Find a ROI which contains the point of the lightning bolt

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

In [30]:
```python
# 2. ROI
point = lightningbolt[150:175, 150:175]
plt.imshow(point, cmap = 'gray')
```

Out[30]: <matplotlib.image.AxesImage at 0x1ffc38f0d90>

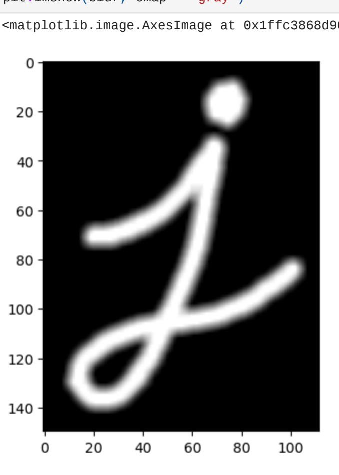


3. Use an averaging kernel on the letter j

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

In [21]:
```python
# 3.
blur = cv2.blur(letterj,(5,5))
plt.imshow(blur, cmap = 'gray')
```

Out[21]: <matplotlib.image.AxesImage at 0x1ffc3868d90>



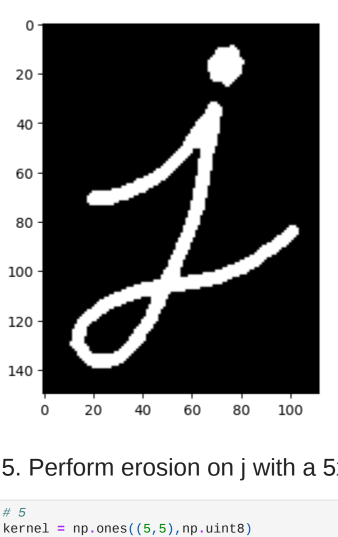
Morphology

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

4. Perform erosion on j with a 3x3 kernel

In [23]:
```python
# 4
kernel = np.ones((3,3),np.uint8)
erosion = cv2.erode(letterj,kernel,iterations = 1)
plt.imshow(erosion, cmap = 'gray')
```

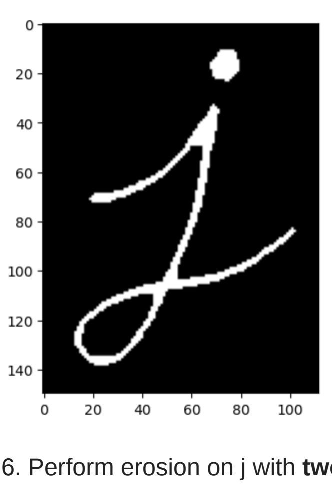
Out[23]: <matplotlib.image.AxesImage at 0x1ffc3831350>



5. Perform erosion on j with a 5x5 kernel

In [24]:
```python
# 5
kernel = np.ones((5,5),np.uint8)
erosion = cv2.erode(letterj,kernel,iterations = 1)
plt.imshow(erosion, cmap = 'gray')
```

Out[24]: <matplotlib.image.AxesImage at 0x1ffc0cc9690>



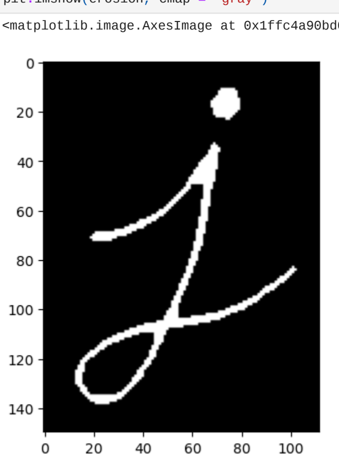
6. Perform erosion on j with **two** iterations, using a kernel size of your choice

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#gaeb1e0c1033e3f6b891a25d0511362aeb

In [26]:
```python
# 6
kernel = np.ones((3,3),np.uint8)
erosion = cv2.erode(letterj,kernel,iterations = 2)
plt.imshow(erosion, cmap = 'gray')
```

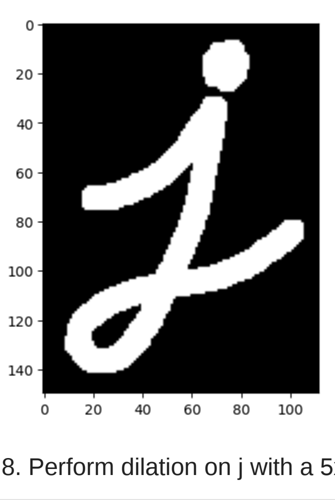
Out[26]: <matplotlib.image.AxesImage at 0x1ffc4a90bd0>



7. Perform dilation on j with a 3x3 kernel

In [27]:
```python
# 7
kernel = np.ones((3,3),np.uint8)
dilation = cv2.dilate(letterj,kernel,iterations = 1)
plt.imshow(dilation, cmap = 'gray')
```

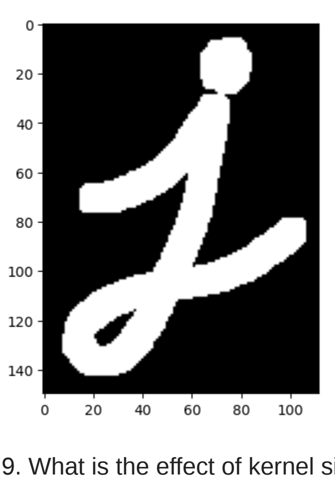
Out[27]: <matplotlib.image.AxesImage at 0x1ffc4afb010>



8. Perform dilation on j with a 5x5 kernel

In [28]:
```python
# 8
kernel = np.ones((5,5),np.uint8)
dilation = cv2.dilate(letterj,kernel,iterations = 1)
plt.imshow(dilation, cmap = 'gray')
```

Out[28]: <matplotlib.image.AxesImage at 0x1ffc4b6b490>



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

In [ ]:
```python
# 9
# Kernal size affects the nature and magnitude of each morphology operation.
```

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 [ ]:
```python
# 10
# Repeated iterations with a small kernal will run the morphology operation multiple times with a smaller matrix while a single iteration with a lar
```

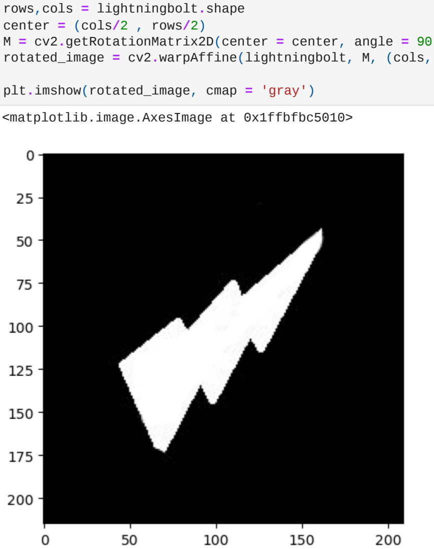
11. Rotate the lightningbolt and star by 90 degrees

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

In [57]:
```python
# 11
rows,cols = lightningbolt.shape
center = (cols/2 , rows/2)
M = cv2.getRotationMatrix2D(center = center, angle = 90, scale = 1)
rotated_image = cv2.warpAffine(lightningbolt, M, (cols, rows))

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

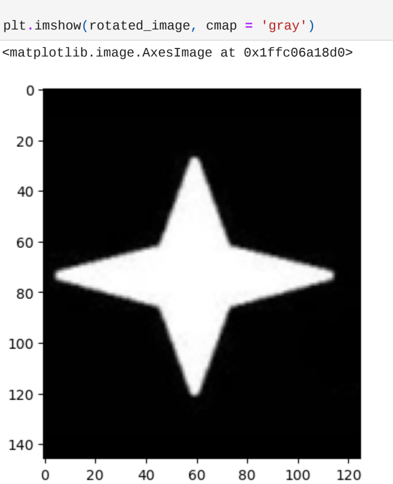
Out[57]: <matplotlib.image.AxesImage at 0x1ffbfbc5010>



In [59]:
```python
rows,cols = star.shape
center = (cols/2 , rows/2)
M = cv2.getRotationMatrix2D(center = center, angle = 90, scale = 1)
rotated_image = cv2.warpAffine(star, M, (cols, rows))

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

Out[59]: <matplotlib.image.AxesImage at 0x1ffc06a18d0>



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`