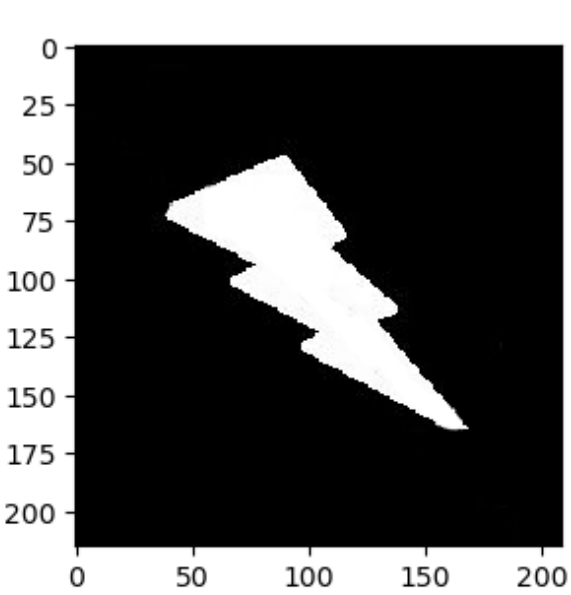
In [1]:
```
%matplotlib inline
from __future__ import print_function
#import ganymede
#ganymede.configure('uav.beaver.works')
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
```

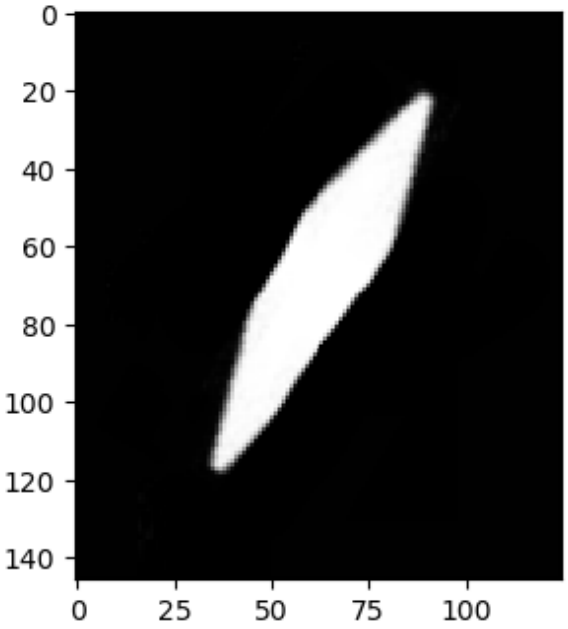
In [ ]:
```
def check(p): pass
check(0)
```

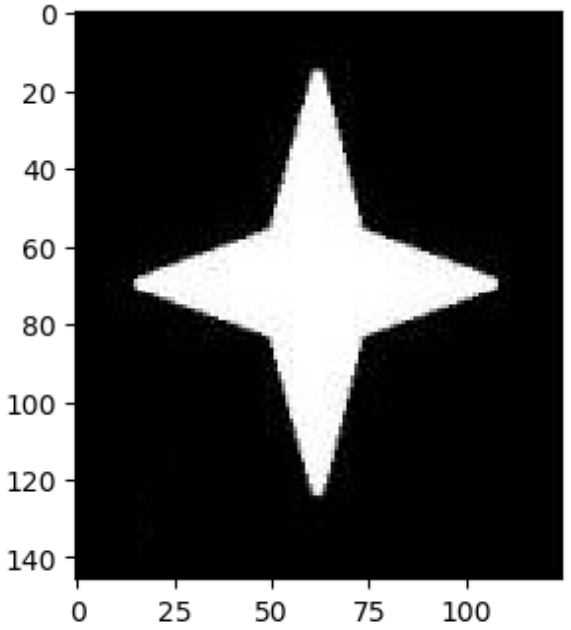
Note

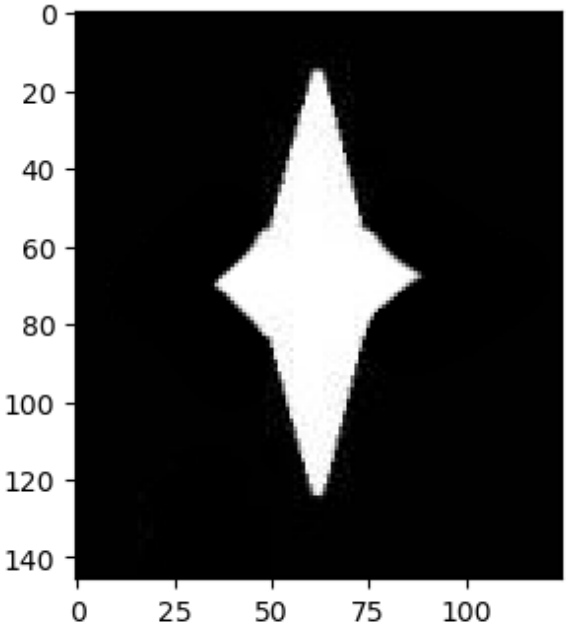
`cv2.imshow()` will not work in a notebook, even though the OpenCV tutorials use it. Instead, use `plt.imshow` and family to visualize your results.

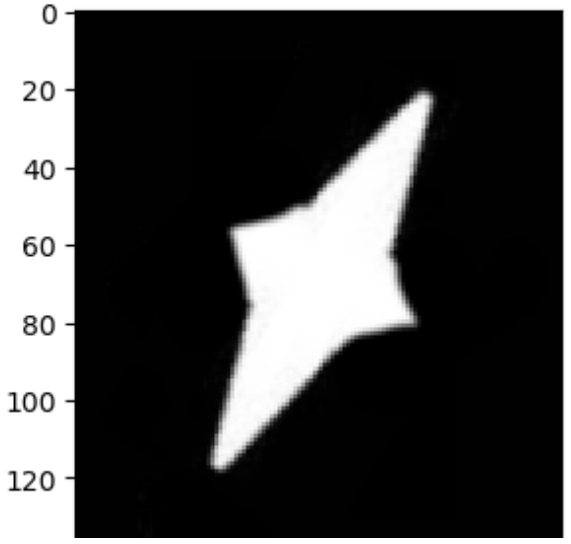
In [17]:
```
#Charlie Lai

lightningbolt      = cv2.imread('shapes/lightningbolt.png', cv2.IMREAD_GR
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_GRA
squishedturnedstar = cv2.imread('shapes/squishedturnedstar.png', cv2.IMRE
letterj            = cv2.imread('shapes/letterj.png', cv2.IMREAD_GRAYSCAL

images = [lightningbolt, blob, star, squishedstar, squishedturnedstar, le

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);
```

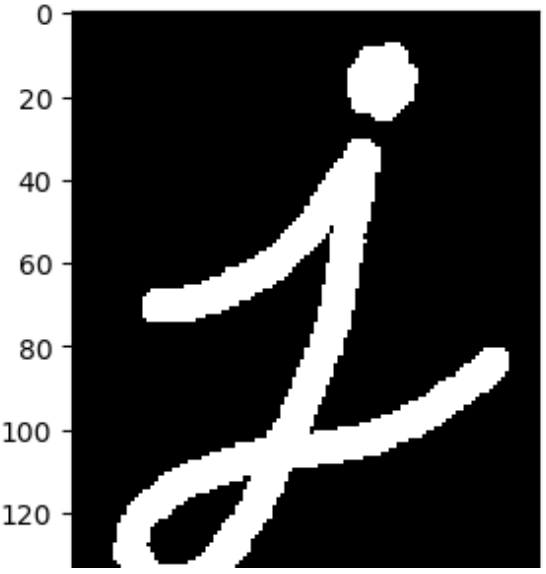

0
25
50
75
100
125
150
175
200
0
50
100
150
200

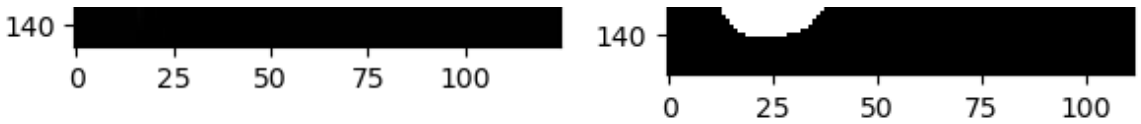

0
20
40
60
80
100
120
140
0
25
50
75
100


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
120


0
20
40
60
80
100
120



In [18]:
```
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?

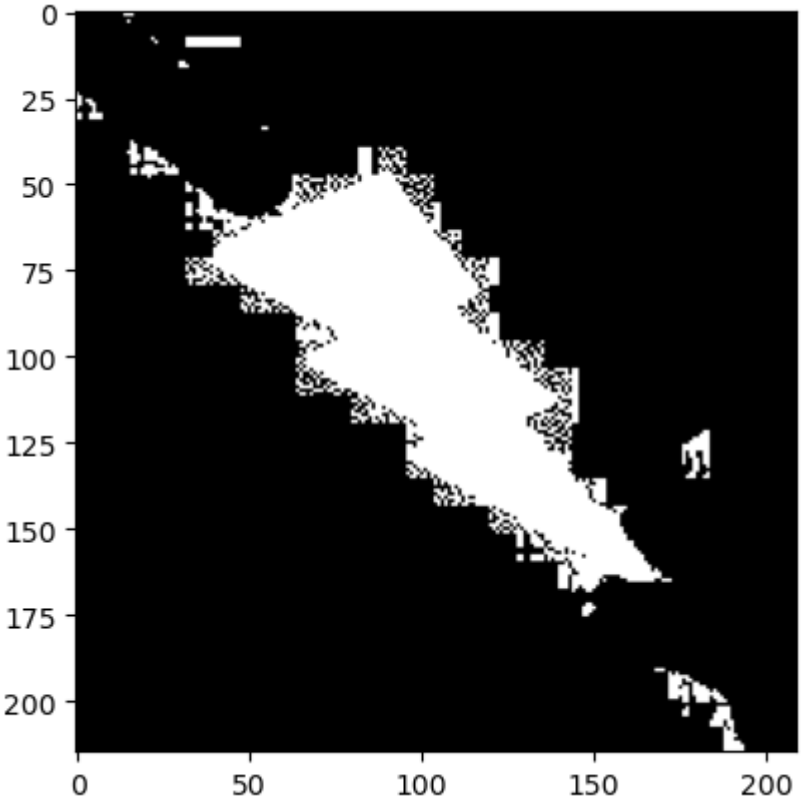
In [ ]:
```
# TODO
# I would expect the value to be 2 because it seems like there are only 2
```

Thresholding

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

In [4]:
```
_, lightningbolt = cv2.threshold(lightningbolt,0,255,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 [ ]:
```
## TODO
## It sets every value that isn't black to white. A more strict value, a
```

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

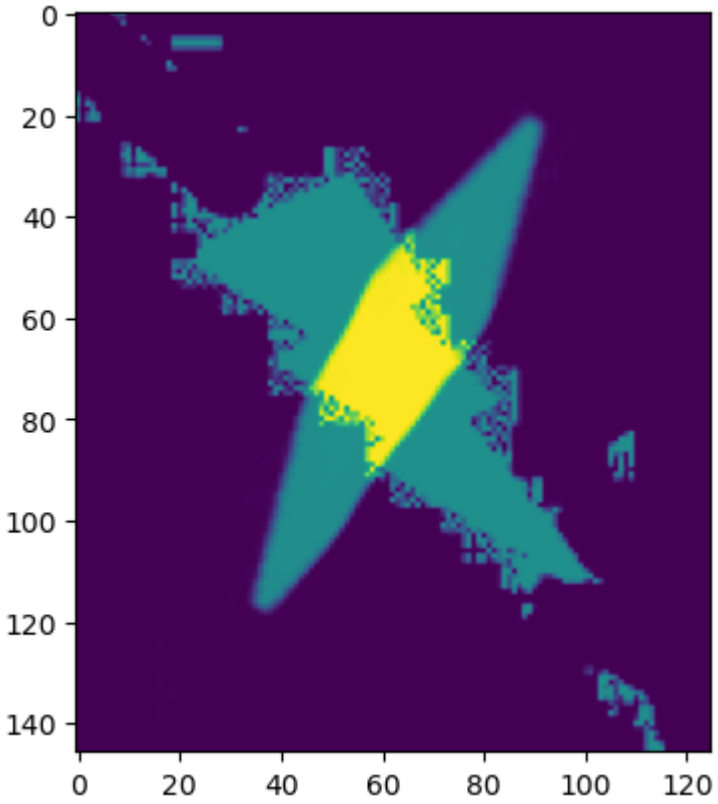
https://docs.opencv.org/3.4.1/d0/d86/tutorial_py_image_arithmetics.html

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

In [5]:
```
# 1. Blend
# TODO
#
height, width = blob.shape[:2]
bolt_resized = cv2.resize(lightningbolt, (width, height), interpolation=c
print(width)
print(height)

img = cv2.addWeighted(bolt_resized, .5, blob, .5, 0)
plt.imshow(img)
```

```
125
146
```

Out[5]:
```
<matplotlib.image.AxesImage at 0xf1173583afe0>
```

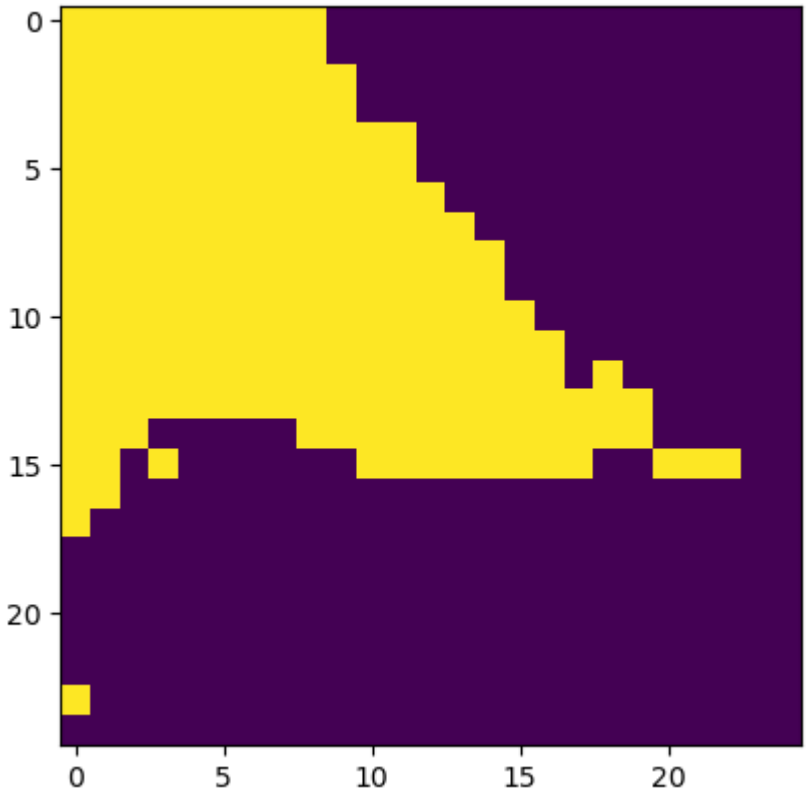


2. Find a ROI which contains the point of the lightning bolt

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

In [6]:
```
# 2. ROI
# TODO

point = lightningbolt[150:175, 150:175]
plt.imshow(point)
```

Out[6]:
```
<matplotlib.image.AxesImage at 0xf11735b85570>
```

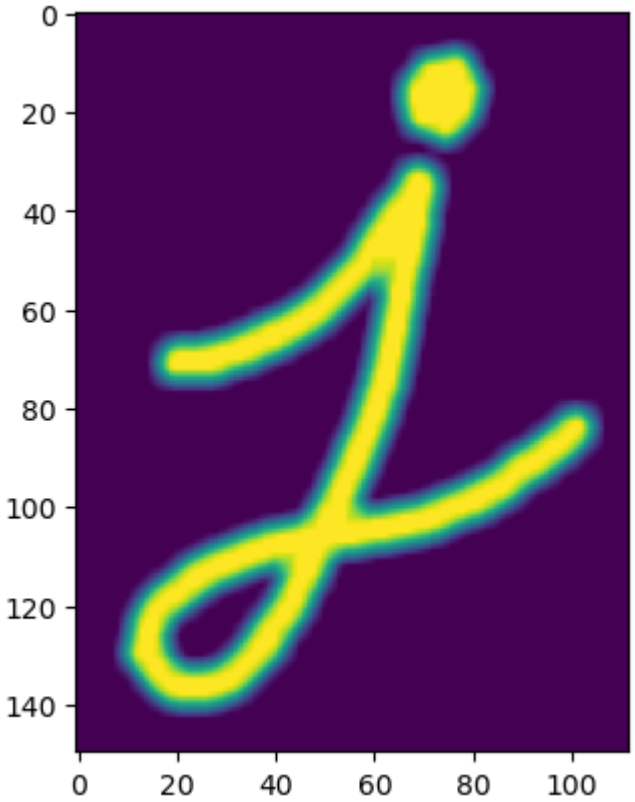


3. Use an averaging kernel on the letter j

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

In [7]:
```
# 3.
# TODO
img = cv2.blur(letterj, (5,5))
plt.imshow(img)
```

Out[7]:
```
<matplotlib.image.AxesImage at 0xf11735bcf5e0>
```



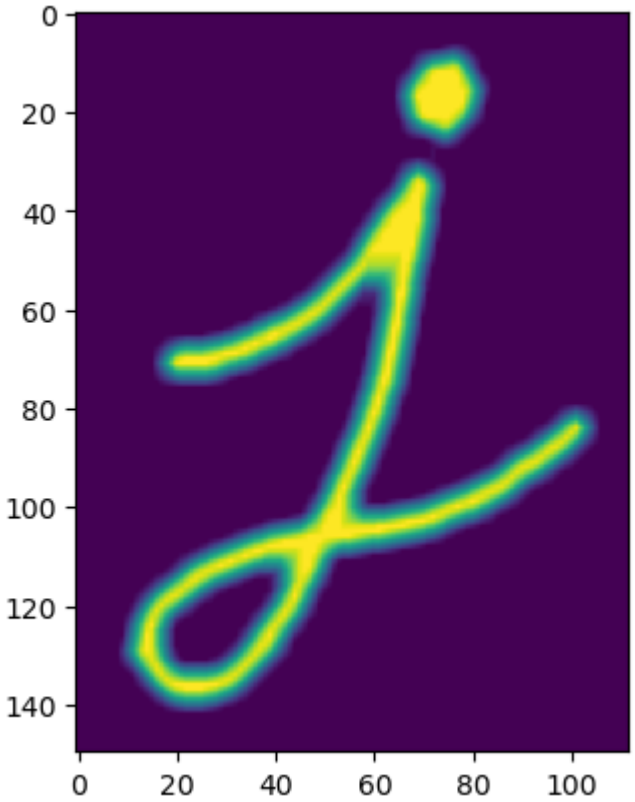
Morphology

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

4. Perform erosion on j with a 3x3 kernel

In [8]:
```
# 4
# TODO
kernel = np.ones((3, 3), np.uint8)
erodedJ = cv2.erode(img, kernel, iterations = 1)
plt.imshow(erodedJ)
```

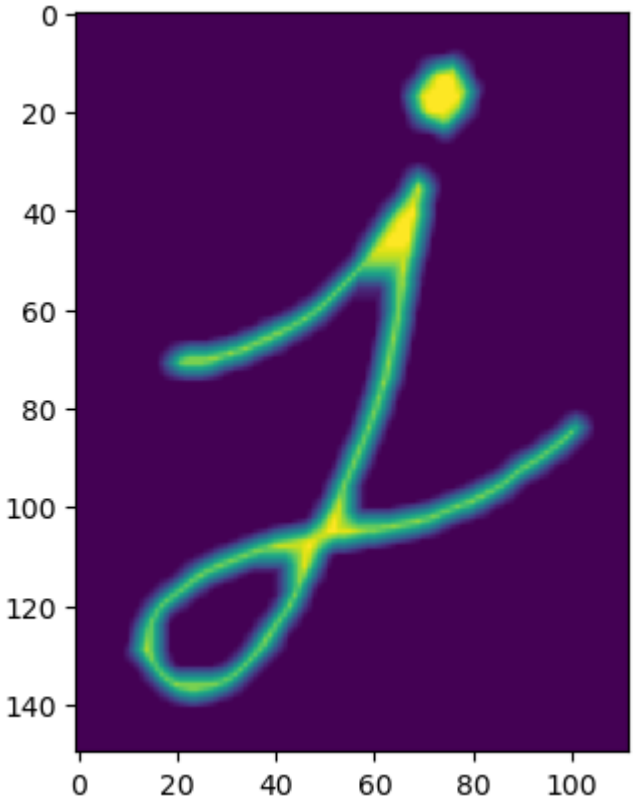
Out[8]:
```
<matplotlib.image.AxesImage at 0xf11735c432b0>
```



5. Perform erosion on j with a 5x5 kernel

In [9]:
```
# 5
# TODO
kernel = np.ones((5, 5), np.uint8)
erodedJ = cv2.erode(img, kernel, iterations = 1)
plt.imshow(erodedJ)
```

Out[9]:
```
<matplotlib.image.AxesImage at 0xf11735aa5de0>
```



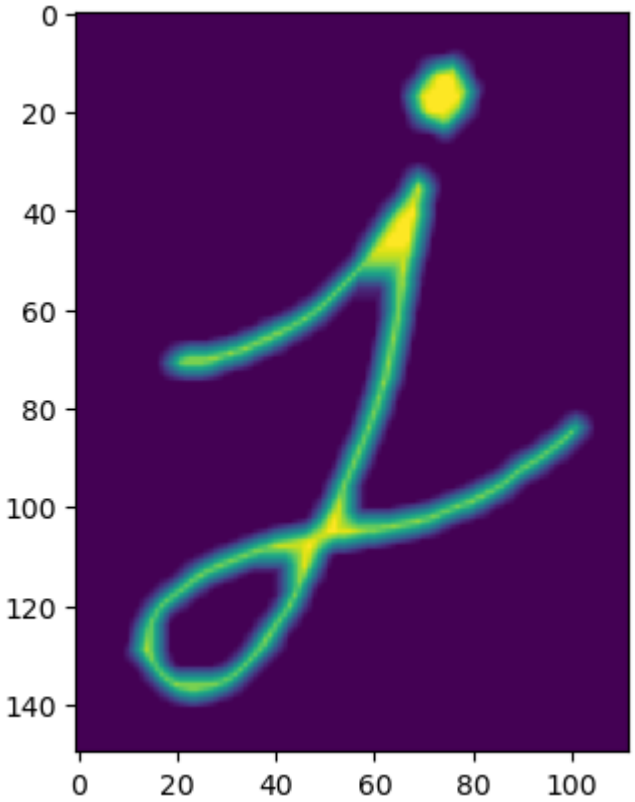
6. Perform erosion on j with **two** iterations, using a kernel size of your choice

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 [10]:
```
# 6
# TODO
kernel = np.ones((3, 3), np.uint8)
erodedJ = cv2.erode(img, kernel, iterations = 2)
plt.imshow(erodedJ)
```

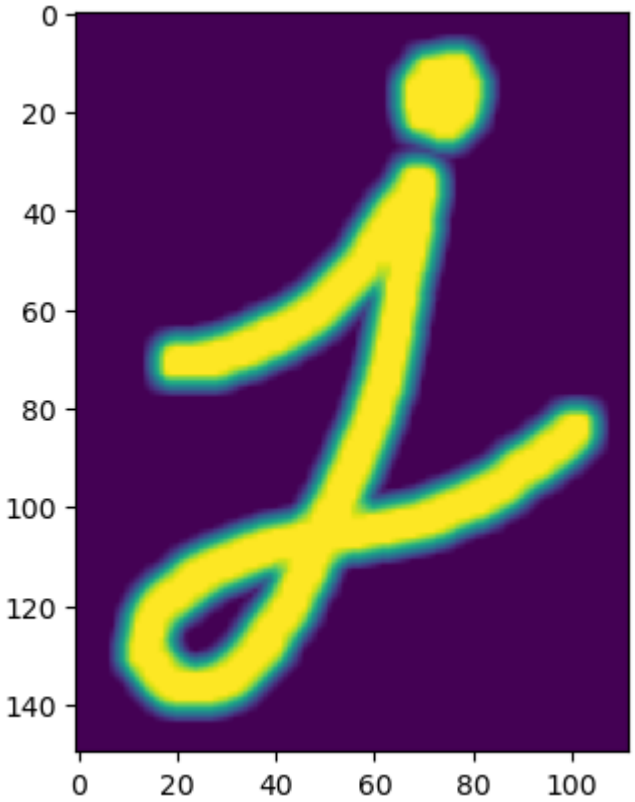
Out[10]:
```
<matplotlib.image.AxesImage at 0xf11735b11720>
```



7. Perform dilation on j with a 3x3 kernel

In [11]:
```
# 7
# TODO
kernel = np.ones((3, 3), np.uint8)
erodedJ = cv2.dilate(img, kernel, iterations = 1)
plt.imshow(erodedJ)
```

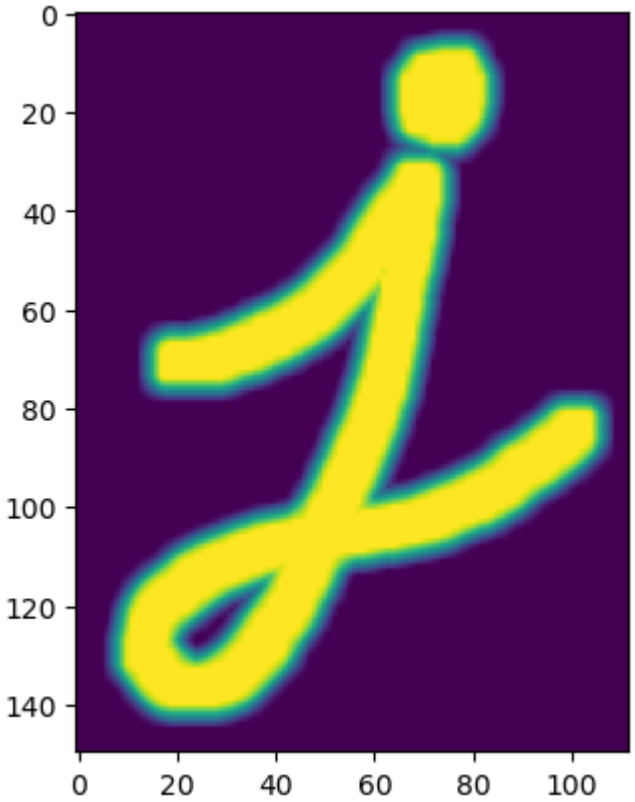
Out[11]:
```
<matplotlib.image.AxesImage at 0xf1173597d150>
```



8. Perform dilation on j with a 5x5 kernel

In [12]:
```
# 8
# TODO
kernel = np.ones((5, 5), np.uint8)
erodedJ = cv2.dilate(img, kernel, iterations = 1)
plt.imshow(erodedJ)
```

Out[12]:
```
<matplotlib.image.AxesImage at 0xf117359f0a60>
```



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

In [13]:
```
# 9
# TODO
# The larger the kernel size, the greater the effect of the morphology op
```

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 [14]:
```
# 10
# TODO
# Repeated iterations with a small kernel would tend to produce gradual a
```

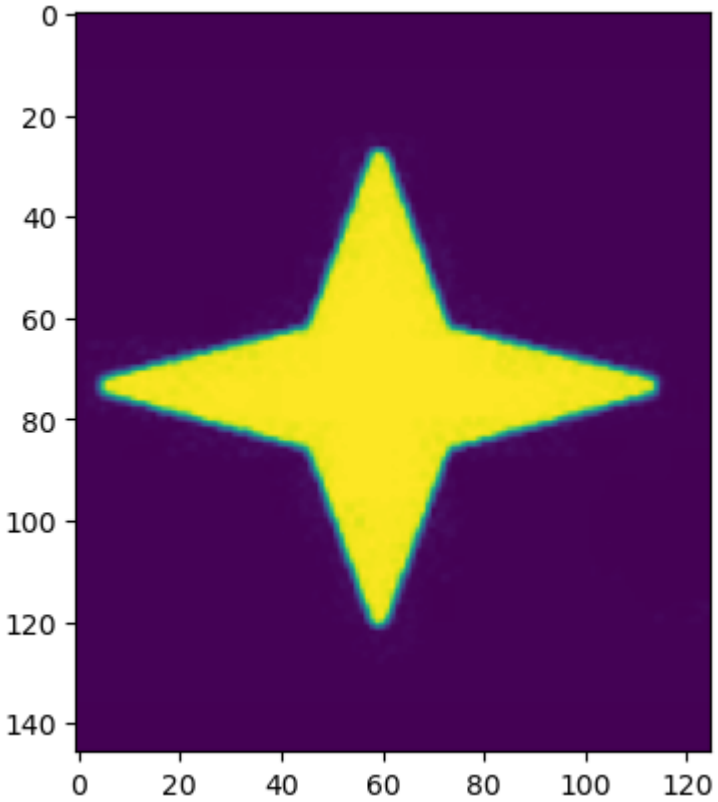
11. Rotate the lightningbolt and star by 90 degrees

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

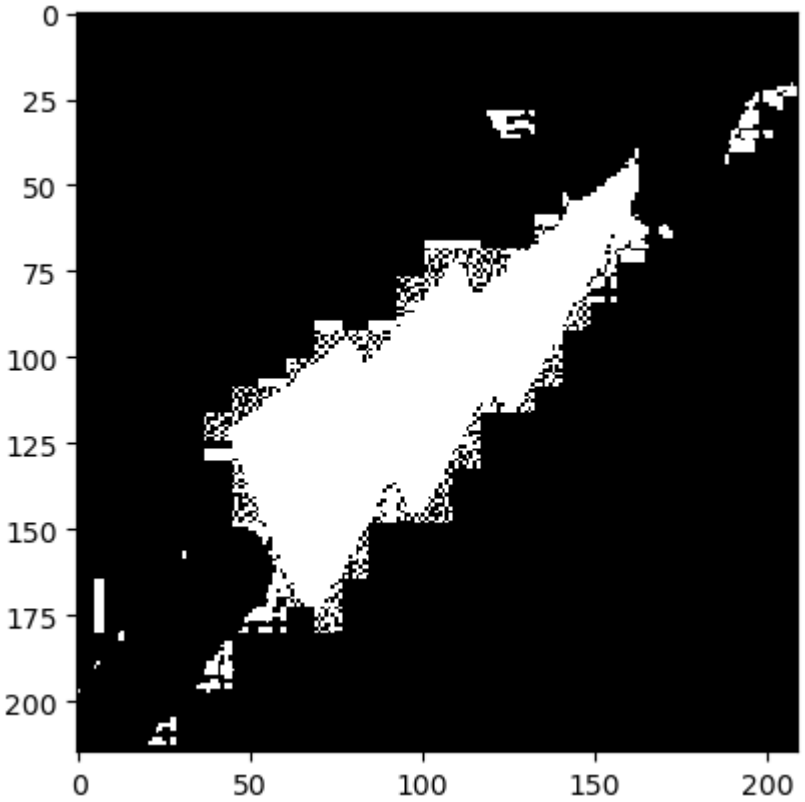
In [15]:
```
# 11
# TODO
rows2,cols2 = star.shape
M2 = cv2.getRotationMatrix2D((cols2/2,rows2/2),90,1)
rotatedstar = cv2.warpAffine(star,M2,(cols2,rows2))
```

```
plt.imshow(rotatedstar)
```

Out[15]: <matplotlib.image.AxesImage at 0xf11735a5d600>



In [16]:
```
rows, cols = lightningbolt.shape

# Create a rotation matrix
M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)

# Apply the affine transformation with cubic interpolation
rotatedbolt = cv2.warpAffine(lightningbolt, M, (cols, rows), flags=cv2.IN

# Display the rotated image
fig, ax = plt.subplots()
ax.imshow(rotatedbolt, cmap='gray', interpolation='none')
plt.show()
```



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`