In [ ]:
```
%matplotlib inline
from __future__ import print_function
import matplotlib.pyplot as plt
import numpy as np
import cv2
import os
```

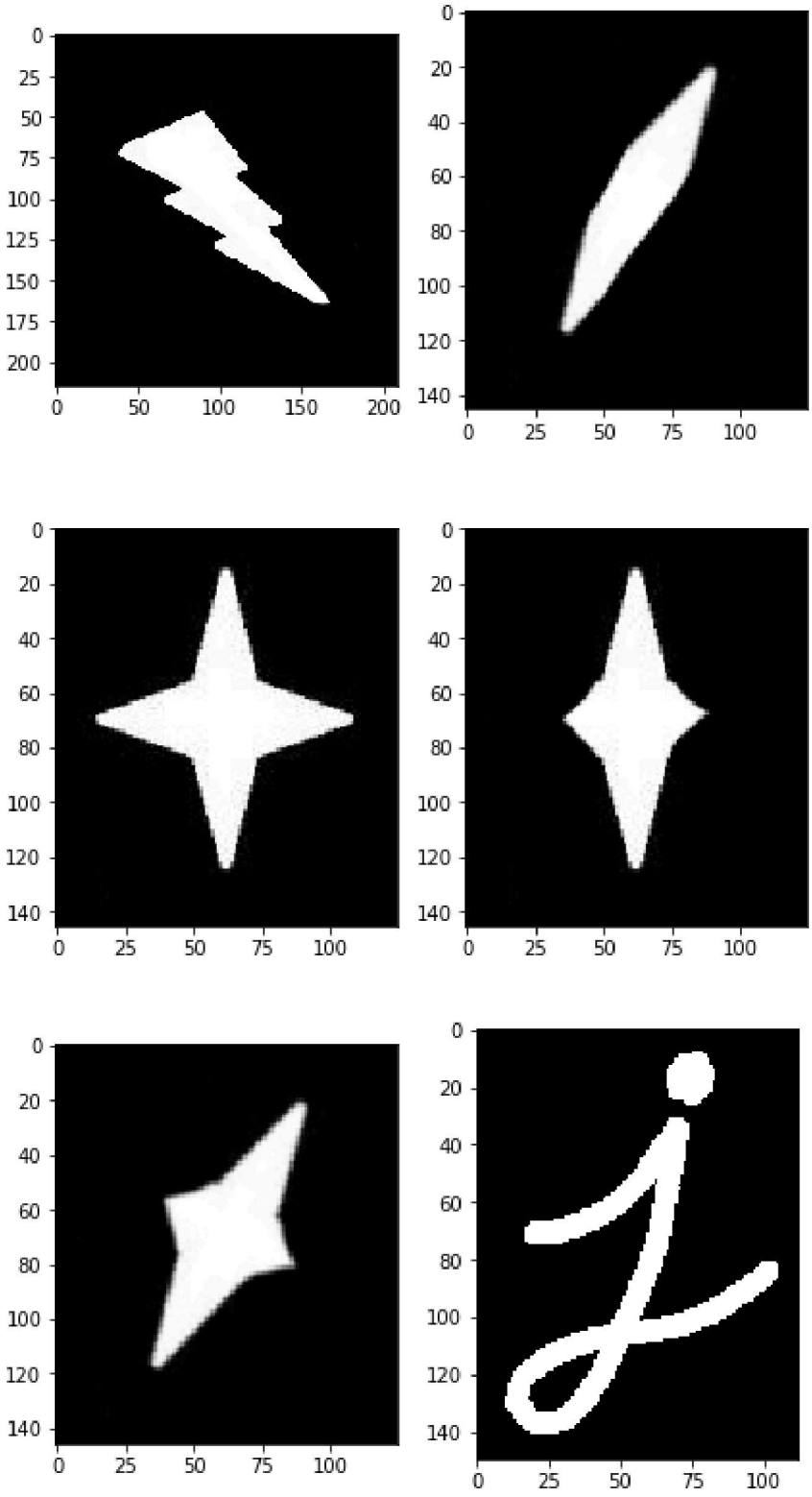
Note

`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 [ ]:
```
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)
squishedbolt       = cv2.imread('shapes/squishedbolt.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 [ ]:
```
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?

You would expect the value the number of pixels in the image; in this case, around 225 x 225. It is only 75 because the set() function only counts unique values, meaning that there are 75 unique intensity values in the array.

Thresholding

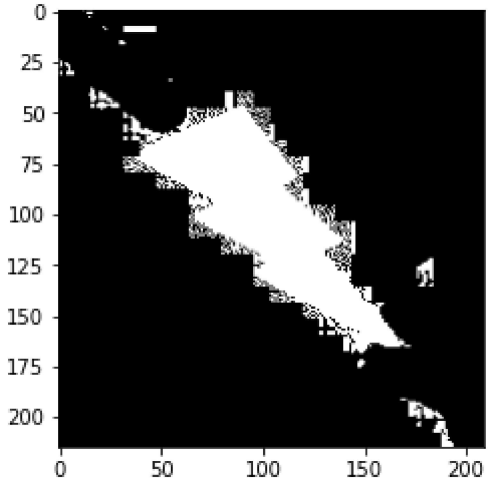
https://docs.opencv.org/3.4.1/d7/d4d/tutorial_py_thresholding.html

In [ ]:
```
_, 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?

When the above values are used for thresholding, it results in all of the even slightly gray pixels becoming white. A better threshold value would be 127 because it is the midpoint on the grayscale; darker pixels will be made black and lighter pixels will be made white.

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

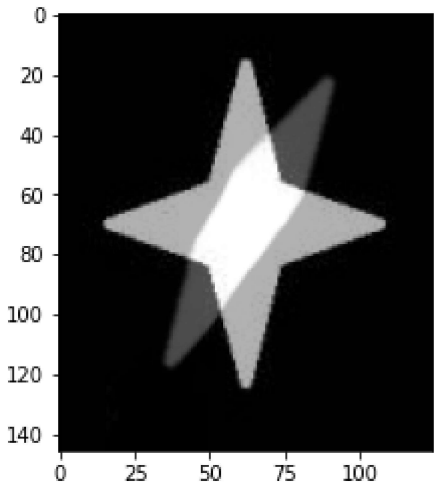
1. Blend letterj and blob together

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 [ ]:
```
dst = cv2.addWeighted(star,0.7,blob,0.3,0)
plt.imshow(dst,cmap='gray')
```

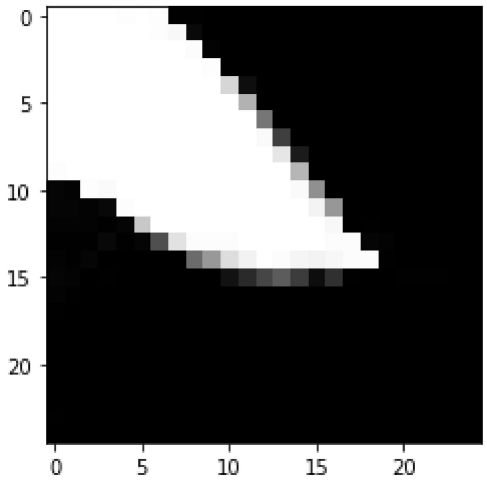
Out[ ]:
```
<matplotlib.image.AxesImage at 0x2165d043220>
```



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

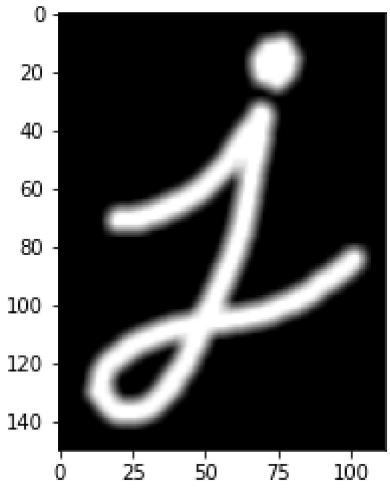
In [ ]:
```
roi = lightningbolt[150:175, 150:175]
plt.imshow(roi,cmap='gray');
```



3. Use an averaging kernel on the letter j

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

In [ ]:
```
kernel = np.ones((5,5),np.float32)/25
avg = cv2.filter2D(letterj,-1,kernel)
plt.imshow(avg,cmap='gray');
```



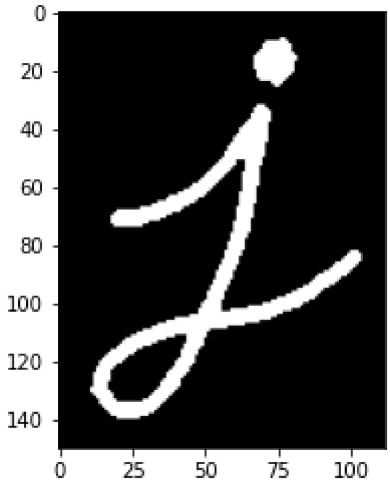
Morphology

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

4. Perform erosion on j with a 3x3 kernel

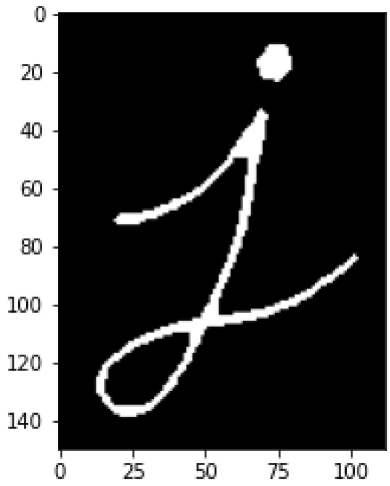
In [ ]:
```
kernel = np.ones((3,3),np.uint8)
erosion_3x3 = cv2.erode(letterj,kernel,iterations = 1)
plt.imshow(erosion_3x3,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x21660049d80>
```



5. Perform erosion on j with a 5x5 kernel

In [ ]:
```
kernel = np.ones((5,5),np.uint8)
erosion_5x5 = cv2.erode(letterj,kernel,iterations = 1)
plt.imshow(erosion_5x5,cmap='gray');
```



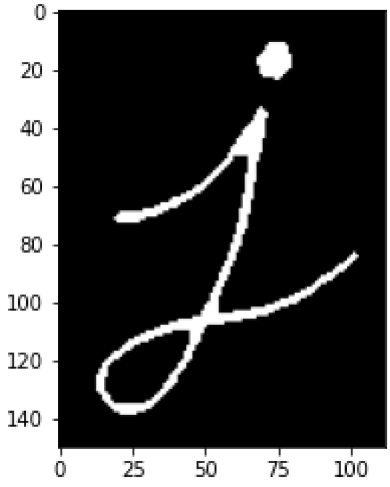
6. Perform erosion on j with **two** iterations, using a kernel size of your choice

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#gaeb1e0c1033e3f6b891a25d051

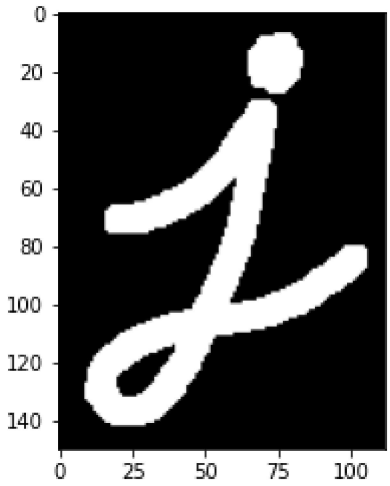
In [ ]:
```
kernel = np.ones((3,3),np.uint8)
erosion_2it = cv2.erode(letterj,kernel,iterations = 2)
plt.imshow(erosion_2it,cmap='gray')
```

Out[ ]:
```
<matplotlib.image.AxesImage at 0x21661d4f490>
```



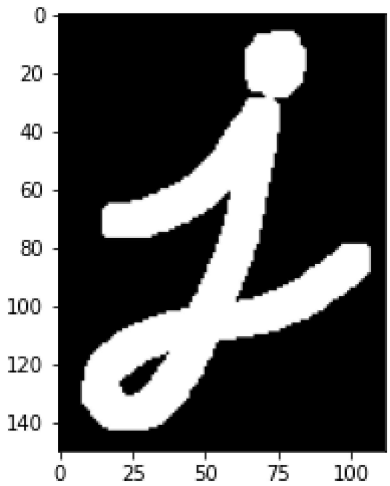
7. Perform dilation on j with a 3x3 kernel

In [ ]:
```
kernel = np.ones((3,3),np.uint8)
dilation_3x3 = cv2.dilate(letterj,kernel,iterations = 1)
plt.imshow(dilation_3x3,cmap='gray');
```



8. Perform dilation on j with a 5x5 kernel

In [ ]:
```
kernel = np.ones((5,5),np.uint8)
dilation_5x5 = cv2.dilate(letterj,kernel,iterations = 1)
plt.imshow(dilation_5x5,cmap='gray');
```



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

The kernel size controls the area that the operation draws information from to perform the operations. For dilation and erosion, this increases the amount the image is dilated or eroded.

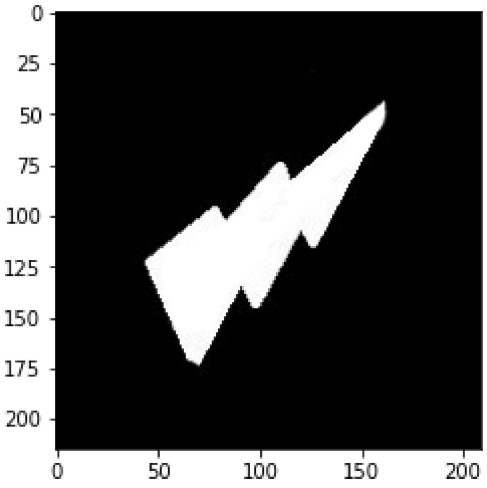
10. What is the difference betweeen repeated iterations of a morphology operation with a small kernel, versus a single iteration with a large kernel?

Repeated iterations of morphology operations with a small kernel will apply the operation multiple times, however the size of the area from which the operation draws information surrounding a pixel stays the same. A single iteration with a large kernel will run the operation one time using information from a large area around the target pixel. These will likely have different end results.

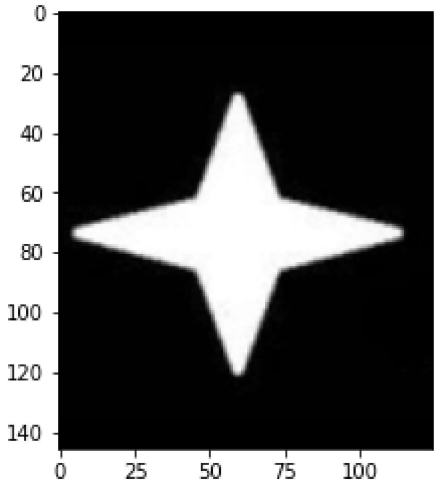
11. Rotate the lightningbolt and star by 90 degrees

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

In [ ]:
```
bolt_rows,bolt_cols = lightningbolt.shape
M_bolt = cv2.getRotationMatrix2D((bolt_cols/2,bolt_rows/2),90,1)
bolt_roated = cv2.warpAffine(lightningbolt,M_bolt,(bolt_cols,bolt_rows))
plt.imshow(bolt_roated,cmap='gray');
```



In [ ]:
```
star_rows,star_cols = star.shape
M_star = cv2.getRotationMatrix2D((star_cols/2,star_rows/2),90,1)
star_roated = cv2.warpAffine(star,M_star,(star_cols,star_rows))
plt.imshow(star_roated,cmap='gray');
```



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