In [ ]: Saanvi Chugh

In [ ]:
```
from __future__ import print_function
import numpy as np
import numpy as np
import matplotlib.pyplot as plt
import os
from IPython.display import HTML, YouTubeVideo
import matplotlib.patches as patches
# Enable inline plots in Jupyter notebook
# import pylatexenc
# !pip install -U matplotlib numpy scipy
```

```
/home/saanvi.local/lib/python3.8/site-packages/matplotlib/projections/__init__.py:63: UserWarning: Unable to import Axes3D. This may be due to multiple versions of Matplotlib being installed (e.g. as a system package and as a pip package). As a result, the 3D projection is not available.
  warnings.warn("Unable to import Axes3D. This may be due to multiple versions of "
```

Enter your name below and run the cell:

Individual cells can be run with {Ctrl} + {Enter}

In [ ]:
```
yournamein name='YOUR NAME HERE'
def checkin:
    print('Checking in as',name)
checkin()
```

https://www.khanacademy.org/math/statistics-probability/describing-relationships-quantitative-data/more-on-regression/squared-error-of-regression-line

Note: All Khan Academy content is available for free at khanacademy.org

In [ ]: YouTubeVideo('6Ovhcf5\7Ic', width=560, height=315)

In [ ]: YouTubeVideo('WC1Ch1Ts7RQ', width=560, height=315)

In [ ]: YouTubeVideo('7ESbouccumo', width=560, height=315)

In [ ]: YouTubeVideo('LcnKuRkPM9g', width=560, height=315)

In [ ]: YouTubeVideo('QRS11Dkkburw', width=560, height=315)

In [ ]: YouTubeVideo('Ewvzcn7bC4g', width=560, height=315)

The last video is optional

In [ ]: YouTubeVideo('ow_yT9cbVhw', width=560, height=315)

In [ ]:
```
lightcircle1 = cv2.imread('shapes/lightcircle1.png', cv2.IMREAD_GRAYSCALE)
_, lightcircle1t = cv2.threshold(lightcircle1, 128,255,cv2.THRESH_BINARY)
# lightcircle1t = 255-lightcircle1t
fig,ax = plt.subplots()
ax.imshow(lightcircle1t, cmap='gray');
print(shape)
```

```
(151, 200)
```

In [ ]:
```
np.argwhere?
Type:        function
Call signature:  np.argwhere(*args, **kwargs)
Type docstring:
Find indices of array elements that are non-zero, grouped by element.

This function works exactly as `np.transpose(np.nonzero(a))`, except that it produces a result of the correct shape for a 0D array.

The output of `argwhere` is not suitable for indexing arrays.
For this purpose use `nonzero(a)` instead.

Parameters
----------
a : array_like
    Input data.

Returns
-------
index_array : (N, a.ndim) ndarray
    Indices of elements that are non-zero. Indices are grouped by element.
    This array will have shape `(N, a.ndim)` where `N` is the number of
    non-zero items.

See Also
--------
where, nonzero

Notes
-----
`np.argwhere(a)` is almost the same as `np.transpose(np.nonzero(a))`, but produces a result of the correct shape for a 0D array.

The output of `argwhere` is not suitable for indexing arrays.
For this purpose use `nonzero(a)` instead.

Examples
--------
>>> x = np.arange(6).reshape(2,3)
>>>
array([[0, 1, 2],
       [3, 4, 5]])
>>> np.argwhere(x>1)
array([[0, 2],
       [1, 0],
       [1, 1],
       [1, 2]])
Class docstring:
Class to wrap functions with checks for __array_function__ overrides.

All arguments are required, and can only be passed by position.

Parameters
----------
dispatcher : function or None
    The dispatcher function that returns a single sequence-like object
    of all arguments relevant. It must have the same signature (except
    the default values) as the actual implementation.
    If `None`, this is a `like`-dispatcher and the
    `like` argument is expected to be the only argument with `like` as the
    default (additional and positional) argument.
implementation : function
    function that implements the operation in NumPy arrays without
    overrides. Arguments passed to this function after the `dispatcher`
    will be forwarded to this (and the `dispatcher`) as if using
    `*args, **kwargs`.

Attributes
----------
_implementation : function
    The original implementation passed in.
```

In [ ]:
```
bw2d = np.argwhere(lightcircle1t)
bw2d
```

Out[ ]:
```
array([[ 47,  88],
       [ 47,  89],
       [ 47,  90],
       ...,
       [104, 109],
       [104, 107],
       [104, 108]])
```

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 [ ]:
```
# TODO
# Your answer here

whitex = bw2d[:, 0:1]
whitex
```

Out[ ]:
```
array([[ 47],
       [ 47],
       [ 47],
       ...,
       [104],
       [104],
       [104]])
```

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

In [ ]:
```
# TODO
# Your answer here
```

To get decimal points, we would need to convert coordinates **from** int values to floats. This would make the values more percise.

In [ ]:
```
def calculate_regression(points):
    '''calculate_regression(points)
    points = points.astype(float) #TODO line estimate, https://docs.scipy.org/doc/numpy/reference/generated/numpy.ndarray.astype.html
    whitex = points[:, 0]
    blackx = points[:, 1]

    xs = whitex.astype(float) #TODO
    ys = blackx.astype(float) #TODO
    x_mean = np.mean(xs) #TODO
    y_mean = np.mean(ys) #TODO

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

    x_squared_mean = np.mean(np.power(xs, 2))

    m = ((x_mean * y_mean) - xy_mean) / ((x_mean ** 2) - x_squared_mean ) #TODO

    b = y_mean - (m * x_mean) #TODO

    return (m,b)

# calculate_regression(bw2d)
```

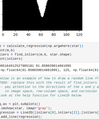
The intercept we calculated, b, may be outside of the pixel space of the image, so we must find two points inside of pixel space, (x_1, y_1) and $(x2, y2)$ which will allow us to plot our regression line on the image. It may be best to choose points on the regression line which also occur on the boundaries/extreme of the image.

In [ ]:
```
def find_intercepts(m, b, shape):
    '''find_intercepts(m, b, shape)'''[1, shape[1]*m + b # TODO
    return (x1, y1, x2, y2)
```

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

_, star = cv2.threshold(star,128,255,cv2.THRESH_BINARY)
# star = 255-star
ax.imshow(star, cmap='gray');
```

```
(149, 149)
```

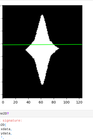


In [ ]:
```
m,b = calculate_regression(np.argwhere(star))
print(m,b)
intercepts = find_intercepts(m,b, star.shape)
print(intercepts)
```

```
0.00114247179016 61.8585304814
(0, np.float64(61.858530481402891), 125, np.float64(61.314144847259))
```

In [ ]:
```
# Below is an example of how to draw a random line from (10,20) to (30,40)
# TODO: replace this with the result of find_intercepts
# Hint: intercepts[0] is the x-coordinate of the first point
#   if shape[1] is the column count, and intercepts[1] is the y-coordinate of
# line on the help function for cv2.line(ax,
# line on the help function for cv2.line(ax,

fig,ax = plt.subplots();
ax.imshow(star, cmap='gray');
regressionline = cv2.line(ax,(0,int(m*0)),(shape[1],int(m*shape[1])),color='lime')
ax.add_line(regressionline);
```



In [ ]:
```
cv2.line?
Type: builtin_function_or_method
Call signature:
cv2.line(
    img,
    pt1,
    pt2,
    color,
    thickness=None,
    lineType=None,
    shift=None,
    pt1=None,
    pt2=None,
    color=None,
    thickness=None,
    lineType=None,
    shift=None,
    offset=None,
    line_type=None,
    dst=None,
    line_type=None,
    shift=None,
    offset=None,
    dst=None,
    line_type=None,
    shift=None,
    offset=None,
    dst=None,
    line_type=None,
    shift=None,
    offset
```

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