In np.random() function there is actually a hidden function np.random.seed(number) where when the number changes the array changes i.e for each array there is correponding seed. Therefore when we specify the seed the random number is generated corresponding to that seed but it dosen't change. This is helpful when we want to again generate same experimental result

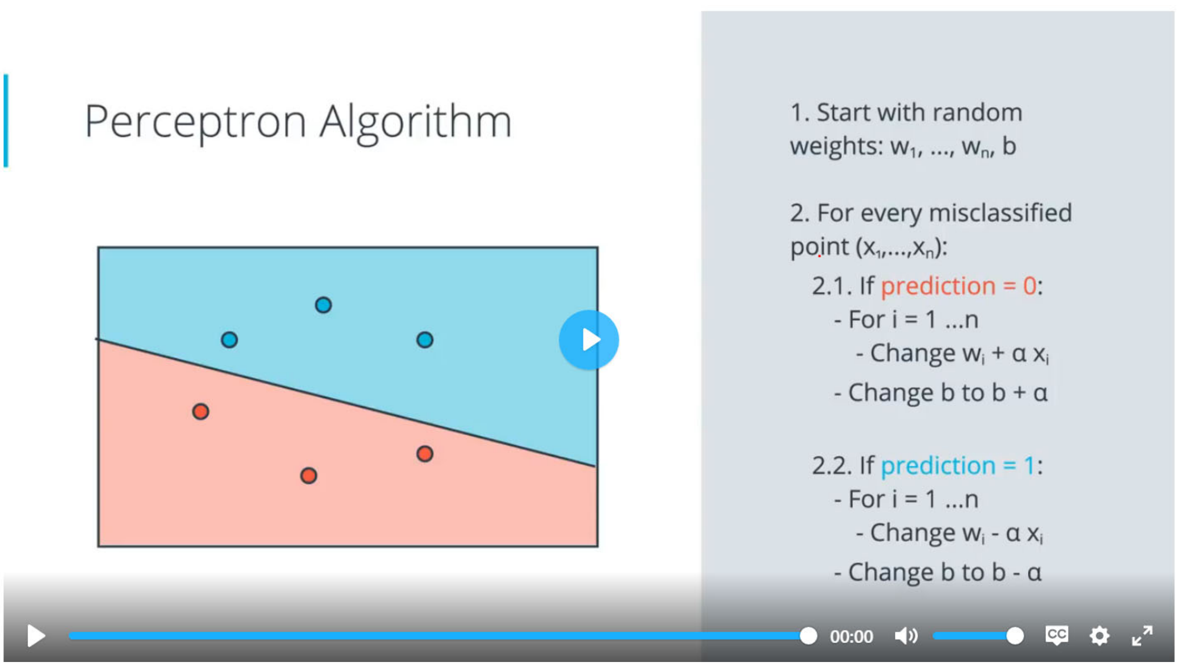
In [2]:
```
import numpy as np

np.random.seed(47)
np_random_array = np.random.randint(10, size=(5,3))
np_random_array
```

Out[2]:
```
array([[7, 6, 7],
       [8, 8, 3],
       [0, 7, 0],
       [7, 7, 1],
       [7, 2, 2]])
```

```
>>> a = np.array([[1, 0],
...               [0, 1]])
>>> b = np.array([[4, 1],
...               [2, 2]])
>>> np.matmul(a, b)
array([[4, 1],
       [2, 2]])
```

Perceptron Agorithm Pseudocode



Recall that the perceptron step works as follows. For a point with coordinates (p, q), label y, and prediction given by the equation $\hat{y} = step(w_1x_1 + w_2x_2 + b)$:

- If the point is correctly classified, do nothing.
- If the point is classified positive, but it has a negative label, subtract $\alpha p, \alpha q,$ and α from $w_1, w_2,$ and b respectively.
- If the point is classified negative, but it has a positive label, add $\alpha p, \alpha q,$ and α to $w_1, w_2,$ and b respectively.

In [3]:

```
import numpy as np
# Setting the random seed, feel free to change it and see different solutions.
np.random.seed(42)

def stepFunction(t):
    if t >= 0:
        return 1
    return 0

def prediction(X, W, b):
    print('prediction X',X,'prediction W',W)
    return stepFunction((np.matmul(X,W)+b)[0])

# Filling in the code below to implement the perceptron trick.
# The function should receive as inputs the data X, the labels y,
# the weights W (as an array), and the bias b,
# update the weights and bias W, b, according to the perceptron algorithm,
# and return W and b.
def perceptronStep(X, y, W, b, learn_rate = 0.01):
    for i in range(len(X)):
        y_hat = prediction(X[i], W, b)
        if y[i] - y_hat == 1:
            W[0] += X[i][0]*learn_rate
            W[1] += X[i][1]*learn_rate
            b += learn_rate
        elif y[i] - y_hat == -1:
            W[0] -= X[i][0]*learn_rate
```

```python
            W[1] -= X[i][1]*learn_rate
            b -= learn_rate
    return W, b

# This function runs the perceptron algorithm repeatedly on the dataset,
# and returns a few of the boundary lines obtained in the iterations,
# for plotting purposes.
# Feel free to play with the learning rate and the num_epochs,
# and see your results plotted below.
def trainPerceptronAlgorithm(X, y, learn_rate = 0.01, num_epochs = 25):
    x_min, x_max = min(X.T[0]), max(X.T[0])
    y_min, y_max = min(X.T[1]), max(X.T[1])
    W = np.array(np.random.rand(2,1))
    b = np.random.rand(1)[0] + x_max
    # These are the solution lines that get plotted below.
    boundary_lines = []
    for i in range(num_epochs):
        # In each epoch, we apply the perceptron step.
        W, b = perceptronStep(X, y, W, b, learn_rate)
        boundary_lines.append((-W[0]/W[1], -b/W[1]))
    return boundary_lines
```

In [4]:

```python
from numpy import genfromtxt
my_data = genfromtxt('Data.csv', delimiter=',')
X = my_data[:,0:2]
y = my_data[:,2]
boundary = trainPerceptronAlgorithm(X,y)
```

```
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
```

```
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
prediction X [ 0.78051  -0.063669] prediction W [[0.37454012]
 [0.95071431]]
```

In [5]:

```
boundary
```

Out[5]:

```
[(array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172])),
 (array([-0.39395654]), array([-1.82178172]))]
```

In [6]:

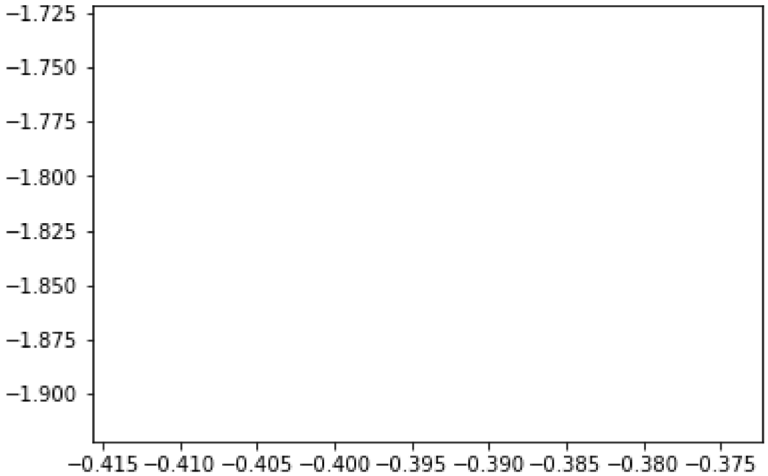
```
x_points = [boundary[i][0] for i in range(len(boundary))]
y_points = [boundary[i][1] for i in range(len(boundary))]
```

In [7]:

```
import matplotlib.pyplot as plt
plt.plot(x_points,y_points)
```

Out[7]:

```
[<matplotlib.lines.Line2D at 0x255e6047220>]
```


−1.725
−1.750
−1.775
−1.800
−1.825
−1.850
−1.875
−1.900
−0.415 −0.410 −0.405 −0.400 −0.395 −0.390 −0.385 −0.380 −0.375