La fonction make_blobs() peut être utilisée pour générer des blobs (objets) de points (avec une distribution Gaussienne). Vous pouvez contrôler le nombre de blobs à générer et le nombre d'échantillons à générer, ainsi qu'une foule d'autres propriétés.

In [90]:
```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```

In [91]:
```
X,y=make_blobs(n_samples=100, n_features=2, centers=2, random_state=0)
```

In [92]:
```
y=y.reshape(y.shape[0], 1)
```

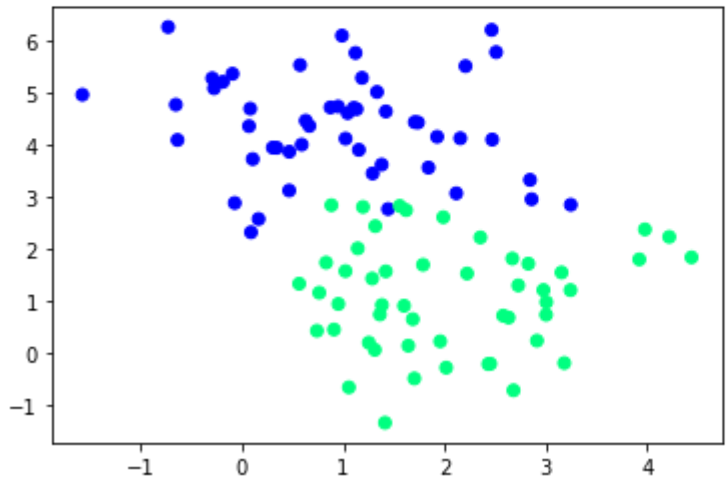
In [93]:
```
print(X.shape)
```

(100, 2)

In [94]:
```
print(y.shape)
```

(100, 1)

In [95]:
```
plt.scatter(X[:,0], X[:,1], c=y, cmap='winter')
plt.show()
```



In [96]:
```
def initialisation (X):
    W=np.random.randn(X.shape[1],1)
    b=np.random.randn(1)
    return (W, b)
```

In [97]:
```
def model(X,W,b):
    Z=X.dot(W)+b
    A=1/(1+np.exp(-Z))
    return A
```

In [98]:
```
def logLoss(A,y):
    return 1/len(y)*np.sum(-y*np.log(A)-(1-y)*np.log(1-A))
```

In [99]:
```
def gradients (A,X,y):
    dW=1/len(y)*np.dot(X.T,A-y)
    db=1/len(y)*sum(A-y)
    return (dW,db)
```

In [100…
```
def update(dW,db, W, b, learning_rate):
    W=W-learning_rate*dW
    b=b-learning_rate*db
    return (W,b)
```

In [110…
```
def prediction(X,W,b):
    A=model(X,W,b)
    print(A)
    return A>=0.5
```

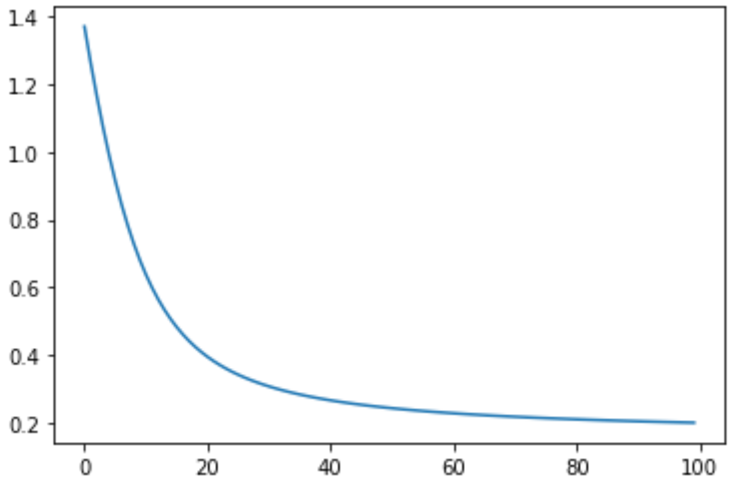
In [111…
```
def artificialNeuron(X,y, learning_rate=0.1, n_iter=100):
    #initialisation
    W,b=initialisation(X)
    cout=[]

    for i in range(n_iter):
        A=model(X,W,b)
        cout.append(logLoss(A,y))
        dW,db=gradients(A,X,y)
        W,b= update(dW,db,W,b,learning_rate)
    y_pred=prediction(X,W,b)
    print(accuracy_score(y, y_pred))
    plt.plot(cout)
    plt.show()
    return (W,b)
```

In [112…
```
W,b=artificialNeuron(X,y)
```

```
[[0.90518696]
 [0.86667506]
 [0.01132461]
 [0.28078781]
 [0.93874719]
 [0.38888957]
 [0.08638289]
 [0.87061309]
 [0.07262944]
 [0.91883116]
 [0.04483846]
 [0.90206045]
 [0.02632409]
 [0.03375207]
 [0.74391498]
 [0.94702976]
 [0.98703282]
 [0.06339483]
 [0.85493212]
 [0.73131689]
 [0.05001392]
 [0.05149766]
 [0.45991368]
```

```
[0.01046351]
[0.94312901]
[0.04009015]
[0.93289607]
[0.00889101]
[0.06444443]
[0.70515219]
[0.97798151]
[0.0511245 ]
[0.79014661]
[0.97813553]
[0.42696328]
[0.17961748]
[0.84181193]
[0.53515844]
[0.42474129]
[0.64086819]
[0.15151145]
[0.03914544]
[0.0020381 ]
[0.18178184]
[0.27725687]
[0.83053896]
[0.97244047]
[0.9726684 ]
[0.01236443]
[0.01263605]
[0.94899951]
[0.71729441]
[0.02326417]
[0.03988819]
[0.69040886]
[0.04773142]
[0.59550494]
[0.86442245]
[0.92604111]
[0.98978021]
[0.71905338]
[0.15137319]
[0.00937775]
[0.9034873 ]
[0.01445113]
[0.39267872]
[0.07027098]
[0.48446086]
[0.90585662]
[0.25149128]
[0.20240142]
[0.94195486]
[0.9270995 ]
[0.09795071]
[0.2341407 ]
[0.06534087]
[0.65371632]
[0.8299762 ]
[0.07739249]
[0.04146875]
[0.33514672]
[0.11849478]
[0.00893206]
[0.10086395]
[0.16759791]
[0.02318333]
```

```
[0.95715244]
[0.04261949]
[0.86655117]
[0.94743512]
[0.86056947]
[0.97401882]
[0.55290042]
[0.96403684]
[0.2221629 ]
[0.00498369]
[0.9873402 ]
[0.9438344 ]
[0.02735878]
[0.57448253]]
```



In [113…

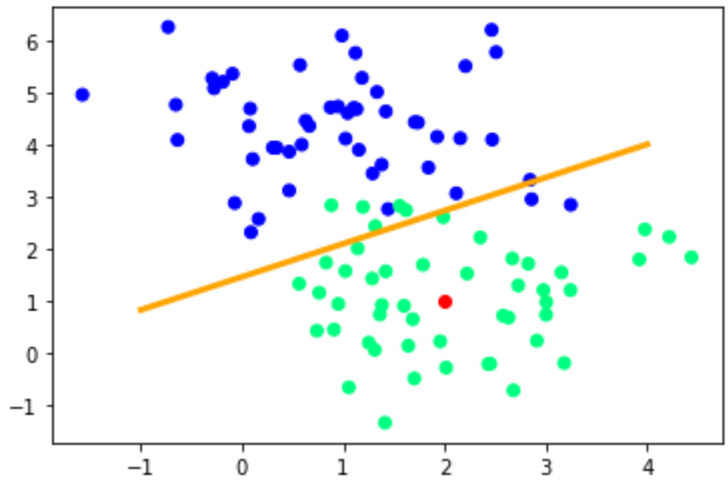
```
nouvellePlante=np.array([2,1])
```

In [117…

```
x0=np.linspace(-1,4,100)
x1=(-W[0]*x0-b)/W[1]
```

In [118…

```
plt.scatter(X[:,0], X[:,1], c=y, cmap='winter')
plt.plot(x0,x1,c='orange',lw=3)
plt.scatter(nouvellePlante[0], nouvellePlante[1],c='r')
plt.show()
```



In [120…

```
prediction(nouvellePlante, W,b)
```

```
[0.88615207]
```

Out[120…

```
array([ True])
```

In [ ]: