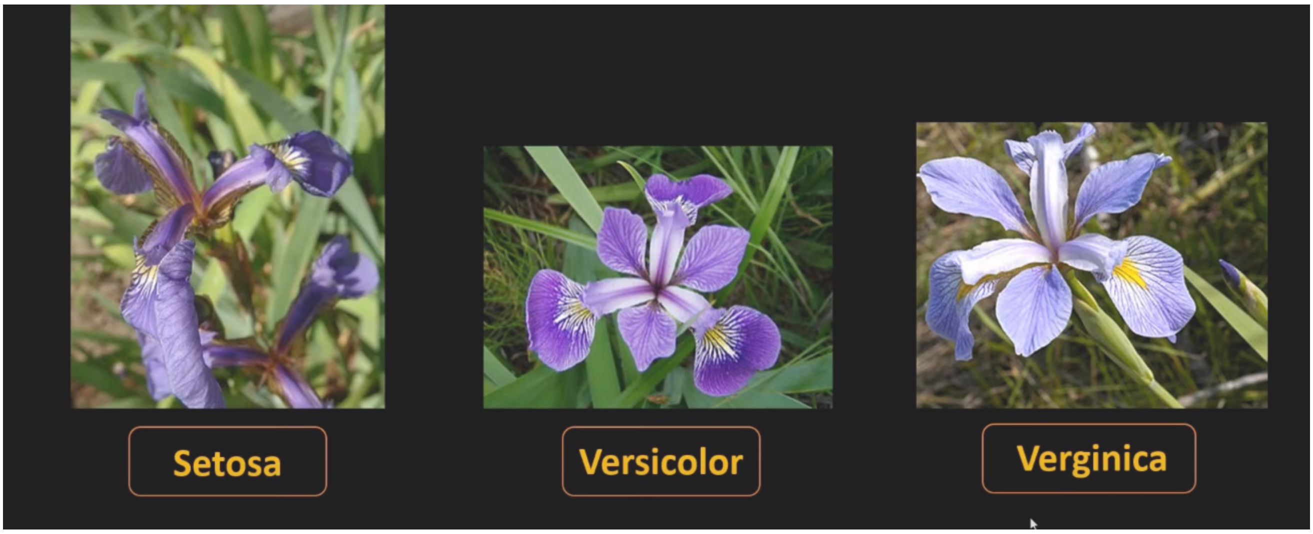
K Nearest Neighbors

Now, lets look into what is **K Nearest Neighbors**.

Lets say you are doing a classification for iris flower dataset.

Here we have a pic of Versicolor flower which is one of the 3 types

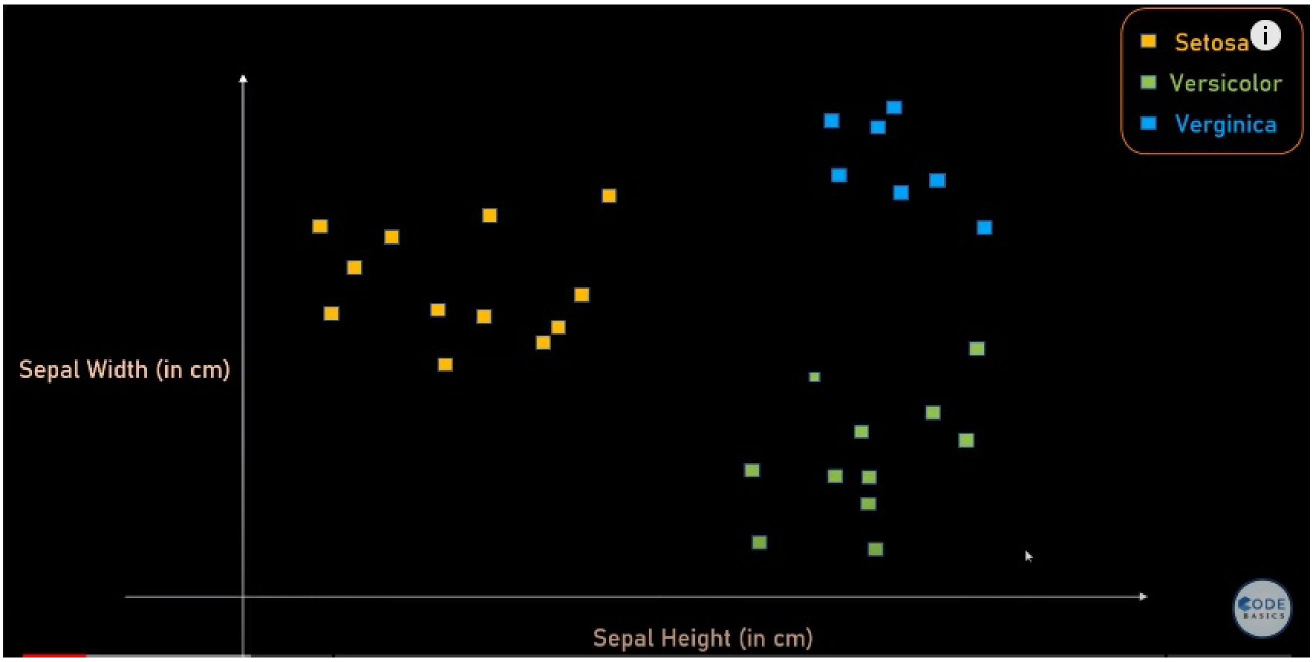
and based on the sepal width amd height u can actually figure out which of the three flowers categories it is in.



So, we are classifying an irish flower into one of the 3 classes :

1. Setosa
2. Versicolor
3. Virginica

You can plot sepal width and height in this kind of 2D scatter plot to figure out which class it belongs to



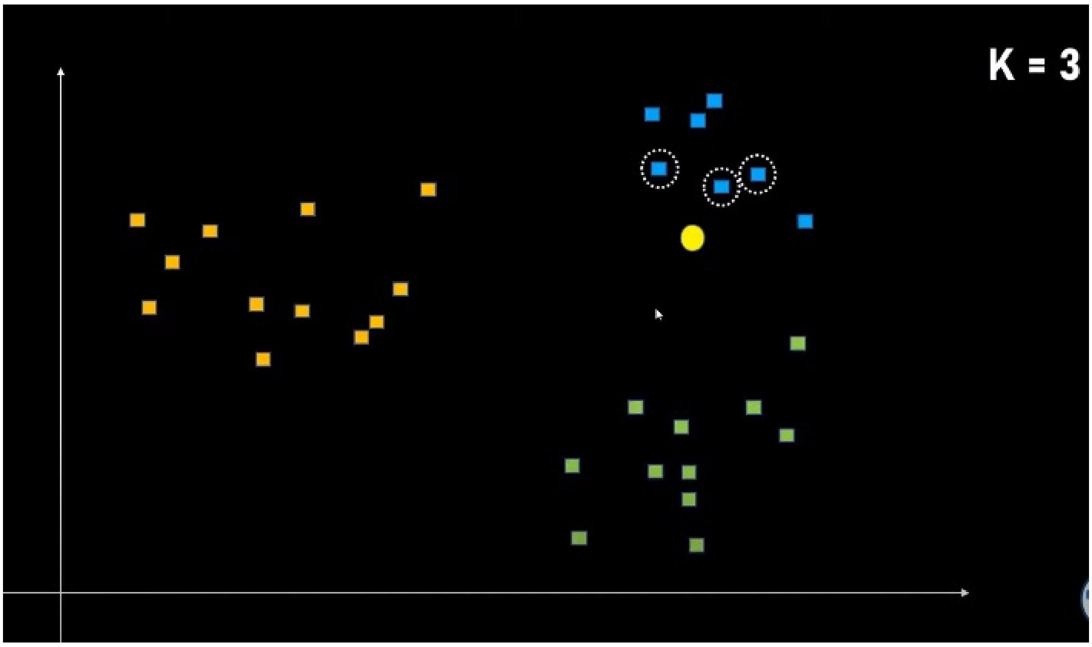
Now, lets say u have build a model and u have a new data point

so the big yellow data point which class it belongs to and using **KNN** you want to figure it out,

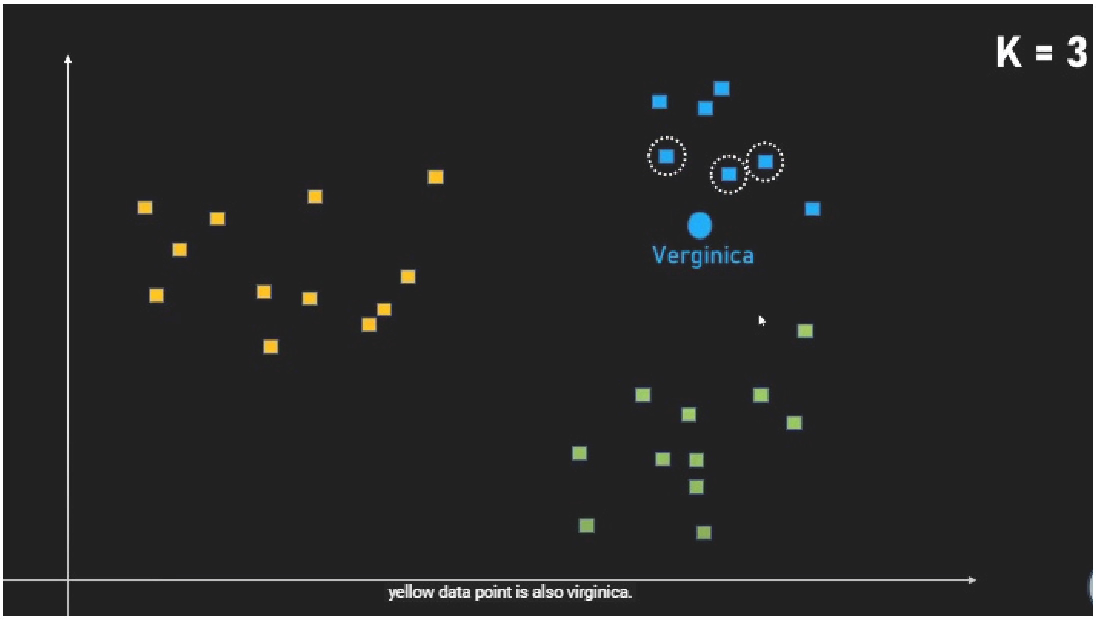
By loking at the graph itself, u can get an idea that this has to be blue color which is **Virginica** cuz it is more near to that Cluster and **KNN** Works just like that.

In **KNN** u first figure out what is the value of **K** and u can figure out the value of **K** by trial and error, there is no specific rule to it, usually people use **5** but u can change it.

So here lets say we use **K=3** and this means we need to figure out the most nearby 3 data points

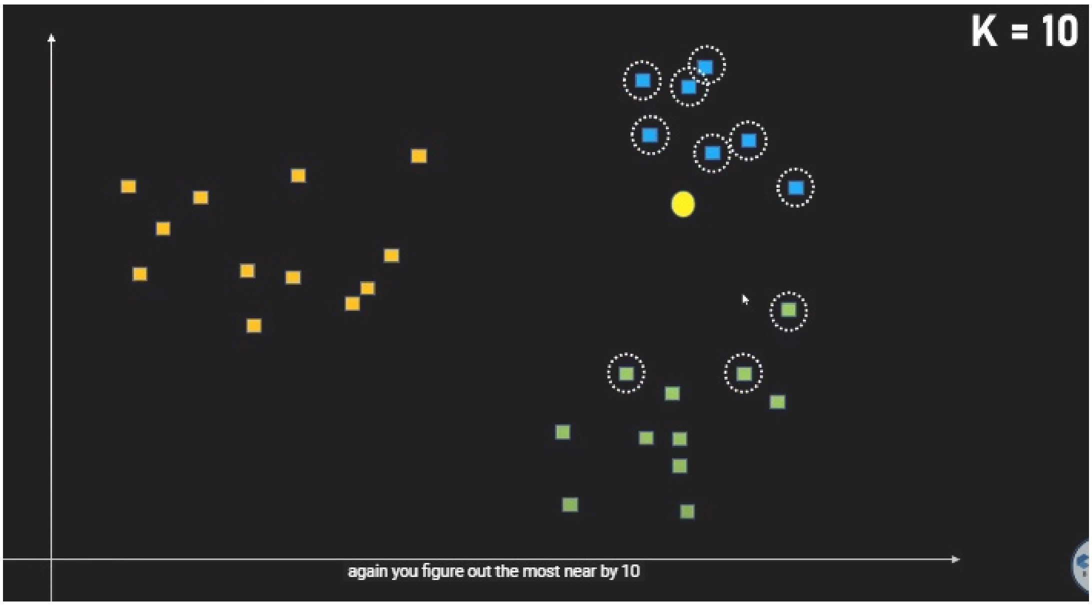


So the most nearby datapoints using the euclidean distance, which is just a simple distance between those two points (big yellow and blue point) are those 3 points (the 3 blue points) and since these 3 are **Virginica** then our yellow data points is also **Virginica**

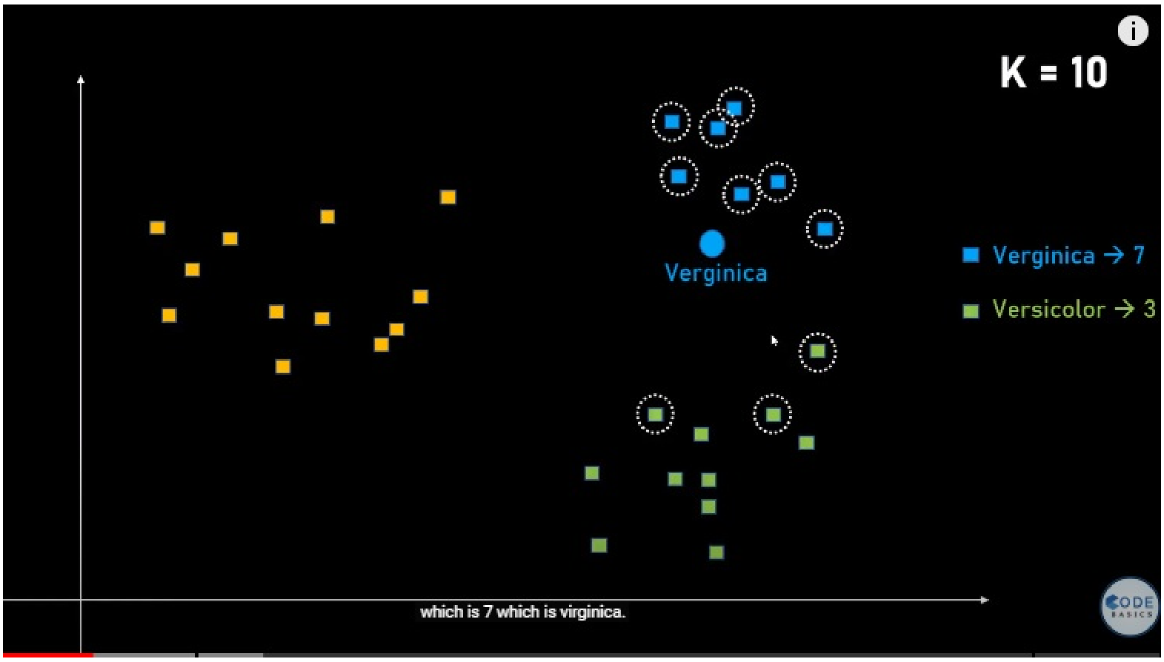


K Nearest Neighbors Algorithm is super simple, You figure out the most nearby **K** datapoint and whichever datapoint category those data points belomg, then our datapoint will also belong to that Category.

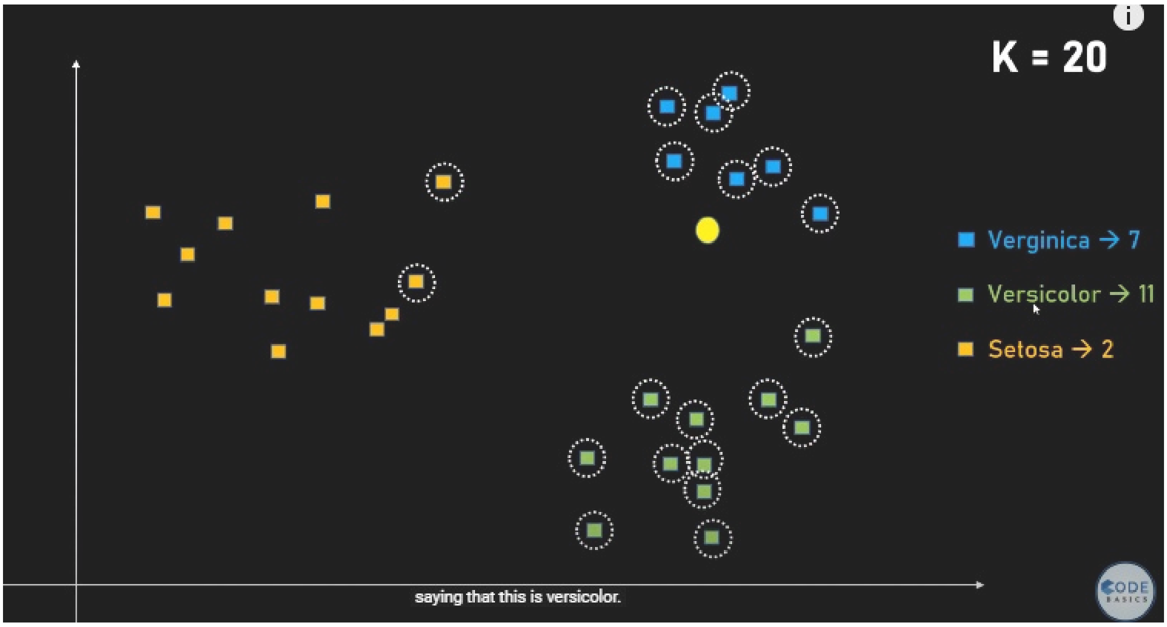
Lets say our **K=10** and u figure out the most nearby 10 datapoints



So here we have 7 Virginica and 3 Versicolor, u just take the Highest number which is 7 means **Virginica**



Now, How about **K=20**, well then we will have a problem because your total number of data point in **Virginica** class are very less so now it is misclassifying this yellow datapoint, saying that it is **Versicolor**



Because the rule is u figureout the most nearby K data point and u take the maximum count, So here K is 20 and the max datapoint which are near is **Versicolor** which is 11.

Well, this will be wrong cuz we know the Yellow datapoint is actually **Virginica** so you have to carefully chose the value of **K** that is not very high nor very low.

One thing we will like to clarify is that here we had only 2 features which is sepel width and height but **KNN** works equally if there are more than two features, we know that there could be n number of features and **KNN** will just work fine

Coding Part

In [1]:
```
import pandas as pd
from sklearn.datasets import load_iris
```

In [2]:
```
data = load_iris()
```

Making our data into a Dataframe

In [4]:
```
df = pd.DataFrame(data.data, columns=data.feature_names)
df.head()
```

Out[4]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

Appending target column to our dataframe

In [7]:

```
df['target'] = data.target
df.head()
```

Out[7]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

Viewing the dataframe with each specific flower classes

In [10]:

```
df[df.target == 1].head()
```

Out[10]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
50	7.0	3.2	4.7	1.4	1
51	6.4	3.2	4.5	1.5	1
52	6.9	3.1	4.9	1.5	1
53	5.5	2.3	4.0	1.3	1
54	6.5	2.8	4.6	1.5	1

In [11]:

```
df[df.target == 2].head()
```

Out[11]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
100	6.3	3.3	6.0	2.5	2
101	5.8	2.7	5.1	1.9	2
102	7.1	3.0	5.9	2.1	2
103	6.3	2.9	5.6	1.8	2
104	6.5	3.0	5.8	2.2	2

In [13]:

```
df[df.target == 0].head()
```

Out[13]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

In [14]:

```
df.shape
```

Out[14]:

```
(150, 5)
```

Split data into 3 dataframe for each flower class

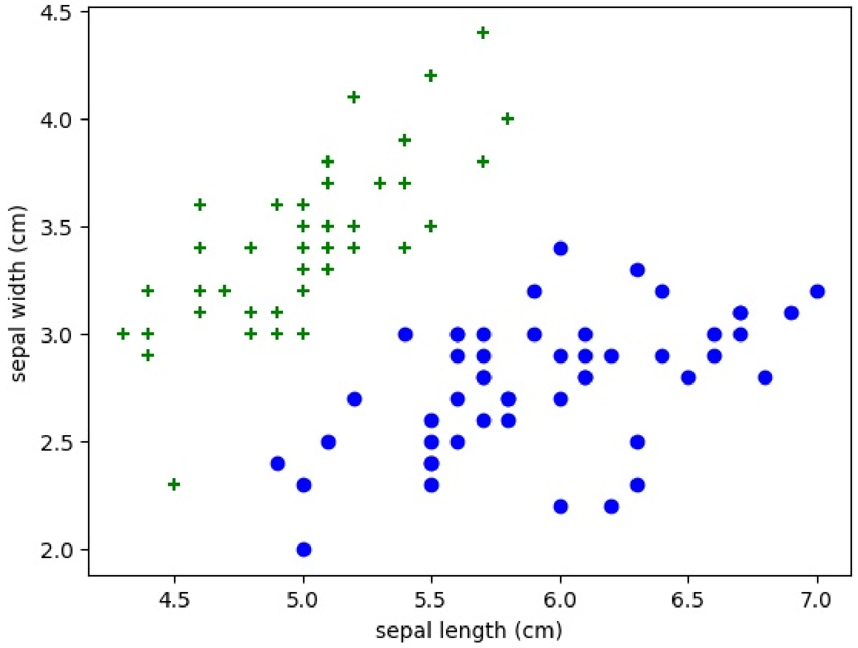
In [15]:

```
df0 = df[:50]
df1 = df[50:100]
df2 = df[100:]
```

Plotting Sepal width and height

In [23]:
```
import matplotlib.pyplot as plt
plt.scatter(df0['sepal length (cm)'], df0['sepal width (cm)'], color="green", marker="+")
plt.scatter(df1['sepal length (cm)'], df1['sepal width (cm)'], color="blue")
plt.xlabel('sepal length (cm)')
plt.ylabel('sepal width (cm)')
```

Out[23]: Text(0, 0.5, 'sepal width (cm)')



Train Test splitting

In [37]:
```
from sklearn.model_selection import train_test_split
X = df.drop(['target'], axis=1)
y = df.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
```

In [38]:
```
len(X_train)
```

Out[38]: 120

In [39]:
```
len(X_test)
```

Out[39]: 30

KNN Algorithm

In [40]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [44]:
```
knn = KNeighborsClassifier(n_neighbors=3) # n_neighbors is the value of K, u can use other params too
```

In [45]:
```
knn.fit(X_train, y_train)
```

Out[45]:
```
▾ KNeighborsClassifier
KNeighborsClassifier(n_neighbors=3)
```

In [46]:
```
knn.score(X_test, y_test)
```

Out[46]: 0.9333333333333333

You can try using different value of **K** to get a better performance, u can use **gridSearchCV** or **K fold cross validation** to figure out the best value of **K**

Confusion Matrix

In [47]:
```
from sklearn.metrics import confusion_matrix
```

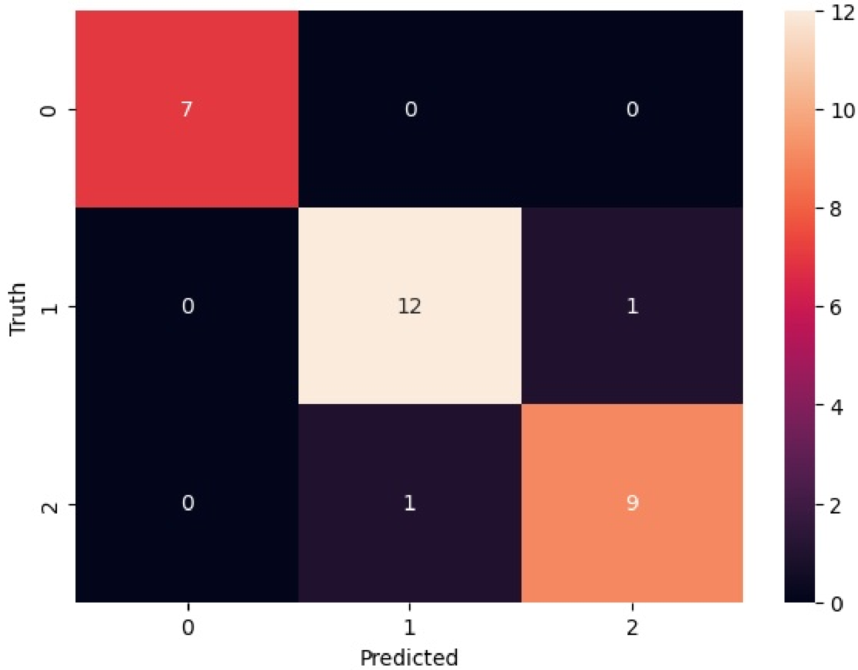
In [47]:
```
from sklearn.metrics import confusion_matrix
```

In [48]:
```
predict = knn.predict(X_test)
cm = confusion_matrix(y_test, predict)
cm
```

Out[48]:
```
array([[ 7,  0,  0],
       [ 0, 12,  1],
       [ 0,  1,  9]], dtype=int64)
```

In [49]:
```
import seaborn as sn
plt.figure(figsize=(7,5))
sn.heatmap(cm, annot=True)
plt.xlabel('Predicted')
plt.ylabel('Truth')
```

Out[49]:
```
Text(58.222222222222214, 0.5, 'Truth')
```



Classification Report

Print classification report for precesion, recall and f1-score for each classes

In [51]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, predict))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       0.92      0.92      0.92        13
           2       0.90      0.90      0.90        10

    accuracy                           0.93        30
   macro avg       0.94      0.94      0.94        30
weighted avg       0.93      0.93      0.93        30
```

Search codebasics Precision recall video to understand what is it