EXPLANATION FOR THE FIRST CODE GIVEN IN THE BOOK

In [9]:

```
from numpy import *
import operator

def createDataSet():                         #THIS IS THE DATASET CREATOR FUNCTION
        group = array([[1.0,1.1],[1.0,1.0],[0,0],[0,0.1]])
        labels = ['A','A','B','B']
        return group, labels
```

In [23]:

```
group,labels = createDataSet()
```

In [33]:

```
print(group,labels,sep='\n\n\n')
```

```
[[1.  1.1]
 [1.  1. ]
 [0.  0. ]
 [0.  0.1]]


['A', 'A', 'B', 'B']
```

In [19]:

```
inX = [1.1,1.2]                                # UNLABBED INPUT
dataSetSize = len(labels)
print(inX,dataSetSize)
```

```
[1.1, 1.2] 4
```

In [34]:

```
group                 # Existing dataset
```

Out[34]:

```
array([[1. , 1.1],
       [1. , 1. ],
       [0. , 0. ],
       [0. , 0.1]])
```

In [31]:

```
tile(inX,(4,1))
# What u want inX to look like. Tile will do the replication for us
```

Out[31]:

```
array([[1.1, 1.2],
       [1.1, 1.2],
       [1.1, 1.2],
       [1.1, 1.2]])
```

In [35]:

```
diffMat=tile(inX, (dataSetSize,1)) - group  # dataSetSize = 4 in this case
diffMat
```

Out[35]:

```
array([[0.1, 0.1],
       [0.1, 0.2],
       [1.1, 1.2],
       [1.1, 1.1]])
```

In [40]:

```
sqDiffMat = diffMat ** 2
sqDiffMat
```

Out[40]:

```
array([[0.01, 0.01],
       [0.01, 0.04],
       [1.21, 1.44],
       [1.21, 1.21]])
```

In [45]:

```
sqDistances = sqDiffMat.sum(axis=1)
sqDistances                  #Squared Euclidean distances array
```

Out[45]:

```
array([0.02, 0.05, 2.65, 2.42])
```

In [47]:

```
distances = sqDistances**0.5
distances             #Actual euclidean distances after taking square root
```

Out[47]:

```
array([0.14142136, 0.2236068 , 1.62788206, 1.55563492])
```

In [64]:

```
# RETURNS THE INDICES WHICH WOULD SORT THE ARRAY IF TRAVESERD FROM 0 to n
sortedDistIndicies = distances.argsort()
sortedDistIndicies
```

Out[64]:

```
array([0, 1, 3, 2], dtype=int32)
```

In [65]:

```
classCount={}
# Empty Dictionary to store how many occurences of a particular label
# are there in the top k data pieces arranged in ascending order of distances
# key: value  ---->  'label' : 'count'
```

In [63]:

```
#vote for the ith label encountered while iterating for the k nearest neighbours
voteIlabel = labels[sortedDistIndicies[2]]
voteIlabel
```

Out[63]:

```
'B'
```

In [66]:

```
classCount={}  #dictionary for count of each label
for i in range(3):                                  #SUPPOSE k=3
    voteIlabel = labels[sortedDistIndicies[i]]      #Find the label for the neighbour
    classCount[voteIlabel] = classCount.get(voteIlabel,0) + 1
    # GET method is used to set default value for each new key to 0

# reverse sort the array according to count of the labels
sortedClassCount = sorted(classCount.items(),key=operator.itemgetter(1), reverse=True)
sortedClassCount[0][0]        #first element in the sorted tuple will be our ans
```

Out[66]:

```
'A'
```