In [1]:
```
print(# /**
#  *                            _ooOoo_
#  *                           o8888888o
#  *                           88" . "88
#  *                           (| -_- |)
#  *                           O\  =  /O
#  *                        ____/`---'\____
#  *                      .'  \\|     |//  `.
#  *                     /  \\|||  :  |||//  \
#  *                    /  _||||| -:- |||||-  \
#  *                    |   | \\\  -  /// |   |
#  *                    | \_|  ''\---/''  |   |
#  *                    \  .-\__  `-`  ___/-. /
#  *                  ___`. .'  /--.--\  `. . __
#  *               ."" '<  `.___\_<|>_/___.'  >'"".
#  *              | | :  `- \`.;`\ _ /`;.`/ - ` : | |
#  *              \  \ `-.   \_ __\ /__ _/   .-` /  /
#  *         ======`-.____`-.___\_____/___.-`____.-'======
#  *                            `=---='
#  *         ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
#  *                      佛祖保佑         永无BUG
#  *         佛曰:
#                   写字楼里写字间，写字间里程序员；
#                   程序人员写程序，又拿程序换酒钱。
#                   酒醒只在网上坐，酒醉还来网下眠；
#                   酒醉酒醒日复日，网上网下年复年。
#                   但愿老死电脑间，不愿鞠躬老板前；
#                   奔驰宝马贵者趣，公交自行程序员。
#                   别人笑我忒疯癫，我笑自己命太贱；
#                   不见满街漂亮妹，哪个归得程序员？
#
)
from math import log
import operator
```

In [2]:
```
from scipy import datasets
def calcSannonEnt(dataSet):
    numEntries=len(dataSet)
    labelConunts={}
    for featVec in dataSet:
        currentlabel=featVec[-1]
        if  currentlabel not in labelConunts.keys():
            labelConunts[currentlabel]=0
        labelConunts[currentlabel]+=1
    shannoonEnt=0
    for key in labelConunts:
        prob=float(labelConunts[key]) / numEntries
        shannoonEnt-=prob*log(prob,2)
    return shannoonEnt

#*****************************************************************

# def calcShannonEnt(dataSet):                          # 计算数据的信息熵(entropy
#     numEntries = len(dataSet)                         # 数据条数
#     labelCounts = {}
#     for featVec in dataSet:
#         currentLabel = featVec[-1]                        # 每行数据的最后一个字
#         if currentLabel not in labelCounts.keys():
```

```
#             labelCounts[currentLabel] = 0                    # 这一步其实就是
#         labelCounts[currentLabel] += 1                      # 统计有多少个类以
#     shannonEnt = 0
#     for key in labelCounts:
#         prob=float(labelCounts[key]) / numEntries           # 计算单个类的熵
#         shannonEnt -= prob*log(prob,2)                      # 累加每个类的熵值
#     return shannonEnt
```

In [3]:
```
# #创建数据集
# def createdataSet1():
#     dataSet = [['长','粗','男'],
#                     ['短','粗','男'],
#                     ['长','细','女'],
#                     ['短','细','女'],
#                     ['短','粗','女'],
#                     ['长','粗','女'],]
#     labels = ['头发','声音'] #特征
#     return dataSet,labels

#************************************************************


def createDataSet1():    # 创造示例数据
    dataSet = [['长', '粗', '男'],
               ['短', '粗', '男'],
               ['短', '粗', '男'],
               ['长', '细', '女'],
               ['短', '细', '女'],
               ['短', '粗', '女'],
               ['长', '粗', '女'],
               ['长', '粗', '女']]
    features = ['头发','声音']  #两个特征
    return dataSet,features
```

In [4]:
```
# #分割数剧集
# def splitdataSet(dataSet,axis,value):
#     retdataSet = []
#     for featVec in dataSet:
#         if featVec[axis]==value:
#             reducefeatVec = featVec[:axis]
#             reducefeatVec.extend(featVec[axis+1:])
#             retdataSet.append(reducefeatVec)
#     return retdataSet

#************************************************************




def splitDataSet(dataSet,axis,value): # 按某个特征分类后的数据
    retDataSet = []
    for featVec in dataSet:
        if featVec[axis] == value:
            reducedFeatVec = featVec[:axis]
            reducedFeatVec.extend(featVec[axis+1:]) # 这两步的操作是没有包括划分
            retDataSet.append(reducedFeatVec)

    return retDataSet
```

In [5]:
```
# #选择最优分类特征
# def chooseBestFeatureToSplit(dataSet):
#     numFeatures = len(dataSet[0])-1
#     baseEntropy = calcSannonEnt(dataSet)
#     bestInfoGain = 0
#     bestFeatture = -1
#     for i in range(numFeatures):
#         featList = [example[i] for example in dataSet]
#         uniqueVals = set(featList)
#         newEntropy = 0
#         for vaule  in uniqueVals:
#           subDataSet = splitdataSet(dataSet,i,vaule)
#           prob = len(subDataSet) / float(len(dataSet))
#           newEntropy+=prob * calcSannonEnt(subDataSet)
#           infoGain = baseEntropy-newEntropy
#         if (bestInfoGain>bestInfoGain):
#             bestFeatture = infoGain
#             bestFeatture = i
#     return bestFeatture

#*****************************************************************************

def chooseBestFeatureToSplit(dataSet):  # 选择最优的分类特征
    numFeatures = len(dataSet[0]) - 1  # 获得特征的个数  2个
    baseEntropy =calcSannonEnt(dataSet)  # 原始的信息熵
    bestInfoGain = 0
    bestFeature = -1
    for i in range(numFeatures): # 遍历两个特征
        featList = [example[i] for example in dataSet]
        uniqueVals = set(featList) # 引入集合
        newEntropy = 0

        for value in uniqueVals:
            subDataSet = splitDataSet(dataSet, i, value) # 根据某个特征分类后的数
            prob = len(subDataSet) / float(len(dataSet))
            newEntropy += prob*calcSannonEnt(subDataSet)  # 按特征分类后的条件经
        infoGain = baseEntropy - newEntropy  # 原始熵与按特征分类后的熵的差值 即
        if (infoGain > bestInfoGain):   # 若按某特征划分后，熵值减少的最大，则次
            bestInfoGain = infoGain
            bestFeature = i
    return bestFeature # 返回的是最优特征的索引
```

In [6]:
```
# def majorityCnt(calssList):
#     classCount = {}
#     for vote in calssList:
#         if vote not in calssList:
#             if vote in classCount.keys():
#                 classCount[vote]=0
#             classCount[vote]+=1
#         sortedclassCount = sorted(classCount.items(),key=operator.itemgetter(1
#         print(sortedclassCount)
#         return sortedclassCount[0][0]

#*****************************************************************************

def majorityCnt(classList):    # 按分类后类别数量排序，比如：最后分类为2男1女，则
    classCount = {}
    for vote in classList:
        if vote not in classCount.keys():
```

```
            classCount[vote] = 0
        classCount[vote] += 1
    sortedClassCount = sorted(classCount.items(), key = operator.itemgetter(1),r
    #print(sortedClassCount)
    return sortedClassCount[0][0]
```

In [7]:
```
# # 错误示范
def createTree(dataSet,labels):
    classList = [example[-1] for example in dataSet]
    if classList.count(classList[0]) == len(classList):
        return classList[0]
    if len(dataSet[0]) == 1:
        return majorityCnt(classList)
    bestFeat = chooseBestFeatureToSplit(dataSet)
    bestFeatlabel = labels[bestFeat]

    myTree = {bestFeatlabel:{}}
    del(labels[bestFeat])
    featValues = [ example[bestFeat] for  example in dataSet]
    uniqueValues = set(featValues)
    for value in uniqueValues:
        subLabels = labels[:]
        myTree[bestFeatlabel] [value] = createTree(splitDataSet(dataSet,bestFeat
    return myTree

#*********************************************************

# 构建决策树(ID3决策树)


# def createTree(dataSet, labels):
#     classList = [example[-1] for example in dataSet]                    # 类别:
#     if classList.count(classList[0]) == len(classList):                 # 最终叶
#         return classList[0]
#     if len(dataSet[0]) == 1:
#         return majorityCnt(classList)
#     bestFeat = chooseBestFeatureToSplit(dataSet)                          #选择最
#     bestFeatLabel = labels[bestFeat]
#     myTree = {bestFeatLabel:{}}                                          # 分类结
#     del(labels[bestFeat])                                                # label
#     featValues = [example[bestFeat] for example in dataSet]
#     uniqueVals = set(featValues)                                          # {'粗
#     for value in uniqueVals:
#         subLabels = labels[:]
#         myTree[bestFeatLabel][value] = createTree(splitDataSet(dataSet, bestFe
#     return myTree
```

In [8]:
```
if __name__=='__main__':
    dataSet, labels = createDataSet1()  # 创造示列数据
    print(createTree(dataSet, labels))  # 输出决策树模型结果
```

```
{'声音': {'细': '女', '粗': {'头发': {'短': '男', '长': '女'}}}}
```