程序代码

In [1]:

```
import pandas as pd
from math import log
from pprint import pprint

log2 = lambda x: log(x, 2)


# We just assume last column as decision column.
_get_decision_col = lambda df: df.columns[-1]


def entropy(df, decision_col=None):
    '''计算数据集的信息熵'''
    decision_col = decision_col or _get_decision_col(df)

    epy = 0.0
    total_items = float(len(df))
    for _, subset in df.groupby(decision_col):
        propotion = len(subset) / total_items

        epy = epy + log2(propotion) * propotion

    return - epy


def gain(df, attribute_col, decision_col=None):
    '''计算数据集根据属性划分后的信息熵'''
    decision_col = decision_col or _get_decision_col(df)

    epy = 0.0
    total_items = float(len(df))

    for _, subset in df.groupby(attribute_col):
        propotion = len(subset) / total_items
        sub_epy = entropy(subset, decision_col)

        epy = epy + propotion * sub_epy

    return epy


def build_decision_tree(df, decision_col=None):
    '''构造决策树'''
    decision_col = decision_col or _get_decision_col(df)

    # Find the attribute with least information gain.
    min_gain, min_attribute_col = float('inf'), None
    for attribute_col in df.columns:
        if attribute_col == decision_col:
            continue
        attribute_gain = gain(df, attribute_col, decision_col)
        if attribute_gain < min_gain:
            min_gain = attribute_gain
            min_attribute_col = attribute_col

    decision = {}
    for attribute_value, subset in df.groupby(min_attribute_col):
        condition = (min_attribute_col, attribute_value)
        epy = entropy(subset, decision_col)
        # No more decision.
        if abs(epy) == 0.0:
            decision[condition] = subset[decision_col].get_values()[0]
        # Calculate sub nodes.
        else:
            decision[condition] = build_decision_tree(subset, decision_col)

    return decision


def explain(tree, decision_action):
    '''输出决策树'''

    def grab_condition(collection, acc, node):
        # decision
        if isinstance(node, str):
            collection.append(acc + [node])
            return

        if not isinstance(node, dict):  # oops
            collection.append(acc)
            return

        for condition, sub_node in node.items():
            grab_condition(collection, acc + [condition], sub_node)

    collection = []
    grab_condition(collection, [], tree)

    for c in collection:
        conditions, decision = c[:-1], c[-1]
        cond_clause = ' and '.join(['{0} = {1}'.format(*c) for c in conditi
ons])
        then_clause = '{0} = {1}'.format(decision_action, decision)

        tmpl = 'if {0} then {1}'.format(cond_clause, then_clause)

        print(tmpl)
```

输入数据

In [3]:
```
data_set = pd.read_csv('data.csv')

data_set
```

Out[3]:

	其他地点	等候条件	周末	顾客	价格	下雨	预约	等候时间估计	等待?
0	yes	no	no	some	ex	no	yes	0-10	yes
1	yes	no	no	full	ch	no	no	30-60	no
2	no	yes	no	some	ch	no	no	0-10	yes
3	yes	no	yes	full	ch	yes	no	10-30	yes
4	yes	no	yes	full	ex	no	yes	>60	no
5	no	yes	no	some	mid	yes	yes	0-10	yes
6	no	yes	no	none	ch	yes	no	0-10	no
7	no	no	no	some	mid	yes	yes	0-10	yes
8	no	yes	yes	full	ch	yes	no	>60	no
9	yes	yes	yes	full	ex	no	yes	10-30	no
10	no	no	no	none	ch	no	no	0-10	no
11	yes	yes	yes	full	ch	no	no	30-60	yes

运行结果

In [4]:
```
explain(build_decision_tree(data_set), data_set.columns[-1])
```

```
if 顾客 = some then 等待? = yes
if 顾客 = full and 价格 = ex then 等待? = no
if 顾客 = full and 价格 = ch and 等候时间估计 = 30-60 and 等候条件 = yes then 等待? =
 yes
if 顾客 = full and 价格 = ch and 等候时间估计 = 30-60 and 等候条件 = no then 等待? =
no
if 顾客 = full and 价格 = ch and 等候时间估计 = >60 then 等待? = no
if 顾客 = full and 价格 = ch and 等候时间估计 = 10-30 then 等待? = yes
if 顾客 = none then 等待? = no
```

In [ ]:
```

```