一、绘制一个棋盘和求解水仙花问题

```
import sys

for i in range(8):
    for j in range(8):
        if (i+j)%2 == 0:
            sys.stdout.write(chr(219))
            sys.stdout.write(chr(219))
        else:
            sys.stdout.write('  ')
    print ('  ')
```

```
#水仙花问题
def jude(n):
    m=n
    a=int(m/100)
    m=m%100
    b=int(m/10)
    m=m%10
    if pow(m,3)+pow(a,3)+pow(b,3)==n:
        return True
    else:
        return False

for i in range(100,1000):
    if jude(i)==True and i!=407:
        print(i,end=',')
    elif i==407:
        print(i)
```

二、python 开发的学生信息管理系统

```
def showInfo():
    '''
        显示可以使用的功能列表给用户
    '''
    print("-"*30)
    print("      学生管理系统    ")
    print(" 1.添加学生的信息")
    print(" 2.删除学生的信息")
    print(" 3.修改学生的信息")
    print(" 4.查询学生的信息")
    print(" 5.遍历所有学生的信息")
    print(" 0.退出系统")
    print( '-'*30)

#定义一个列表，用来存储多个学生的信息
students=[]
```

```python
def addStudent():
    '''
        添加一个学生，需要传入姓名、年龄、学号
    '''
    #输入学员姓名、年龄、学号
    stuName = input("请输入学生姓名：")
    stuId = input("请输入学生学号(学号不可重复)：")
    stuAge = input("请输入学生年龄:")
    #验证学号是否唯一 #i记录要删除的下标，leap为标志位，如果找到leap=1，否则为0
    i = 0
    leap = 0
    #循环判断
    for stu in students:
        if stu['stuId'] == stuId:
            leap = 1
            break
        else:
            i = i + 1
    #leap == 1代表学生学号
    if leap == 1:
        print("输入学生学号重复，添加失败！")
    else:
        # 定义一个字典，存放单个学生信息
        stuInfo = {}
        stuInfo['stuName'] = stuName
        stuInfo['stuId']   = stuId
        stuInfo['stuAge']  = stuAge

        # 单个学生信息放入列表
        students.append(stuInfo)
        print("添加成功！")
```

```python
#删除学生函数
def deleteStudent():
    '''
        根据学号删除学生，学号
    '''
    print("您选择了删除学生功能")
    delId=input("请输入要删除的学生学号:")
    #i记录要删除的下标，leap为标志位，如果找到leap=1，否则为0
    i = 0
    leap = 0
    for stu in students:
        if stu['stuId'] == delId:
            leap = 1
            break
        else:
            i=i+1
    if leap == 0:
        print("没有此学生学号，删除失败！")
    else:
        del students[i]
        print("删除成功！")
```

```python
#修改学生函数，有bug
def updateStudent():
    '''
        根据学号修改学生信息，学号
    '''
    print("您选择了修改学生信息功能")
    alterId=input("请输入你要修改学生的学号:")
    #检测是否有此学号，然后进行修改信息
    i = 0
    leap = 0
    for stu in students:
        if stu['stuId'] == alterId:
            leap = 1
            break
        else:
            i = i + 1
    if leap == 1:
        updateOperate()
    else:
        print("没有此学号，修改失败！")
```

```python
def updateOperate():
    '''
        根据用户选择不同的操作来修改学生的信息
    '''
    while True:
        alterNum=int(input(" 1.修改学号\n 2.修改姓名 \n 3.修改年龄 \n 4.退出修改\n"))
        if alterNum == 1:
            newId=input("输入更改后的学号:")
            #修改后的学号要验证是否唯一
            i = 0
            leap1 = 0
            for stu1 in students:
                if stu1['stuId'] == newId:
                    leap1 = 1
                    break
                else:
                    i = i + 1
            if leap1 == 1:
                print("输入学号不可重复，修改失败！")
            else:
                stu1['stuId']=newId
                print("学号修改成功")
        elif alterNum == 2:      #修改姓名操作
            newName=input("输入更改后的姓名:")
            stu['stuName'] = newName
            print("姓名修改成功")
```

```python
        elif alterNum == 3:      #修改年龄操作
            newAge=input("输入更改后的年龄:")
            stu['stuAge'] = newAge
            print("年龄修改成功")
        elif alterNum == 4:
            break
        else:
            print("输入错误请重新输入")
```

```python
#查询单个学生信息函数
def getStudentById():
    '''
        根据学号查询学生信息,需要传入学号
    '''
    print("您选择了查询学生信息功能")
    searchID=input("请输入你要查询学生的学号:")
    #验证是否有此学号
    i = 0
    leap = 0
    for stu in students:
        if stu['stuId'] == searchID:
            leap = 1
            break
        else:
            i = i + 1
    if leap == 0:
        print("没有此学生学号，查询失败！")
    else:
        print("找到此学生，信息如下：")
        print("学号：%s\n姓名：%s\n年龄：%s\n"%(stu['stuId'],stu['stuName'],stu['stuAge']))
```

```python
#查询所有学生信息函数
def getAllStudent():
    '''
        查询所有学生信息
    '''
    #遍历并输出所有学生的信息
    print('*'*20)
    print("接下来进行遍历所有的学生信息...")
    print("stuId       姓名          年龄")
    for stu in students:
        print("%s      %s      %s"%(stu['stuId'],stu['stuName'],stu['stuAge']))
    print("*"*20)
```

```python
#主函数
def main():
    '''
        主函数：程序的入口
    '''
    while True:
        #把功能列表进行显示给用户
        showInfo()

        #提示用户选择功能
        #获取用户选择的功能
        key = int(input("请选择功能（序号）："))

        #根据用户选择，完成相应功能
        if key == 1:
            addStudent()
        elif key == 2:
            deleteStudent()
        elif key == 3:
            updateStudent()
        elif key == 4:
            getStudentById()
        elif key == 5:
            getAllStudent()
        elif key == 0:
            #退出功能，尽量往不退出的方向引
            quitconfirm = input("亲，真的要退出么 （yes或者no）")
            if quitconfirm == 'yes':
                print("欢迎使用本系统，谢谢")
                break;1
        else:
            print("您输入有误，请重新输入")

main()
```

三、调用 sklearn 库和癌症数据集实现决策树

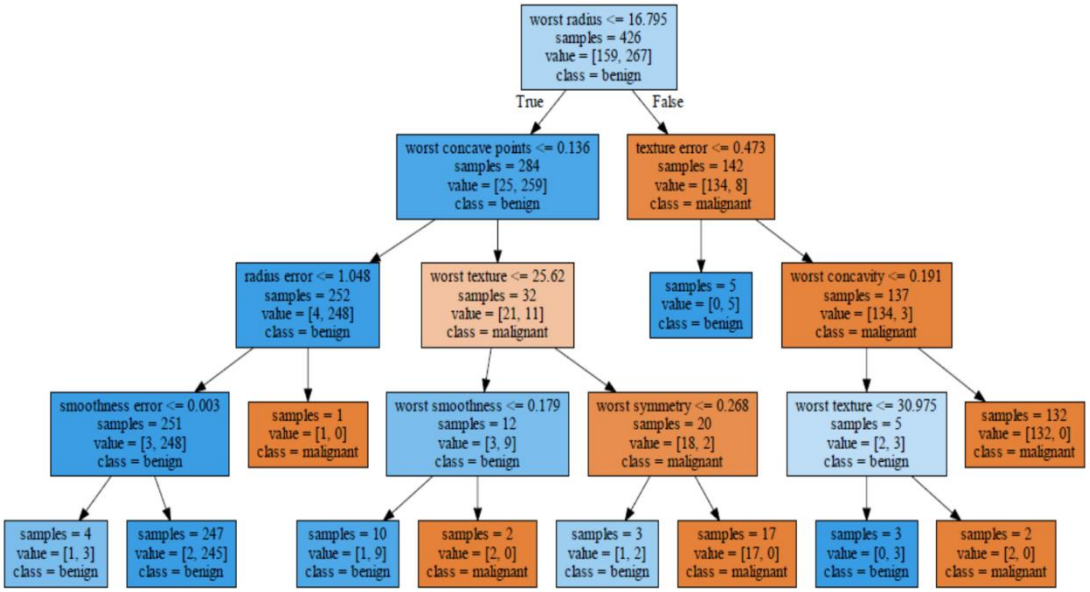
由于要使用 graphviz 函数，需要配置这个库，方法如下：

1）打开 Anaconda Prompt 后，输入 conda install graphviz，然后再输入 pip install graphviz，

2）将 graphviz 路径添加在系统环境变量中，如 D:\Anaconda3\Library\bin\graphviz.

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split

cancer =load_breast_cancer()
X_train,X_test,y_train,y_test = train_test_split(cancer.data,
                                                 cancer.target,
                                                 stratify=cancer.target,
                                                 random_state=42)
tree = DecisionTreeClassifier(random_state=0)
tree.fit(X_train,y_train)
print("Accuracy on training set:{:3f}".format(tree.score(X_train,y_train)))
print("Accuracy on testing set:{:3f}".format(tree.score(X_test,y_test)))

tree = DecisionTreeClassifier(max_depth=4,random_state=0)
tree.fit(X_train,y_train)
print("Accuracy on training set:{:.3f}".format(tree.score(X_train,y_train)))
print("Accuracy on testing set:{:.3f}".format(tree.score(X_test,y_test)))


from sklearn.tree import export_graphviz

export_graphviz(tree,
                out_file="tree.dot",
                class_names=["malignant", "benign"],
                feature_names=cancer.feature_names,
                impurity=False,
                filled=True)

import graphviz
with open("tree.dot") as f:
    dot_graph = f.read()

dot = graphviz.Source(dot_graph)
dot.view()
```

运行效果：

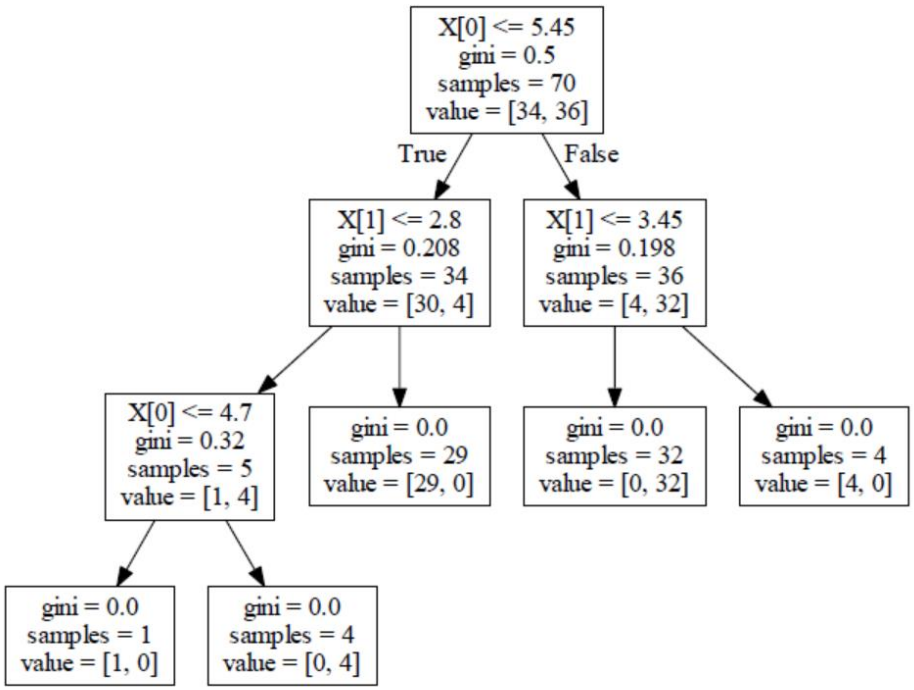


四、调用 sklearn 库和 iris 数据集实现决策树

```
import numpy as np
import pandas as pd
#import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.tree import export_graphviz
import graphviz

# data
def create_data():
    iris = load_iris()
    df = pd.DataFrame(iris.data, columns=iris.feature_names)
    df['label'] = iris.target
    df.columns = ['sepal length', 'sepal width', 'petal length', 'petal width', 'label']
    data = np.array(df.iloc[:100, [0, 1, -1]])
    # print(data)
    return data[:,:2], data[:,-1]

X, y = create_data()
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3)
```

```
'''
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=None,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=None,
            splitter='best')
'''
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train,)

clf.score(X_test, y_test)

tree_pic = export_graphviz(clf, out_file="mytree.pdf")
with open('mytree.pdf') as f:
    dot_graph = f.read()

dot = graphviz.Source(dot_graph)
dot.view()
```

运行效果:



五、课本上的数据集

```
import numpy as np
import pandas as pd
from math import log
from sklearn.tree import export_graphviz

def create_data():
    datasets = [['青年', '否', '否', '一般', '否'],
                ['青年', '否', '否', '好', '否'],
                ['青年', '是', '否', '好', '是'],
                ['青年', '是', '是', '一般', '是'],
                ['青年', '否', '否', '一般', '否'],
                ['中年', '否', '否', '一般', '否'],
                ['中年', '否', '否', '好', '否'],
                ['中年', '是', '是', '好', '是'],
                ['中年', '否', '是', '非常好', '是'],
                ['中年', '否', '是', '非常好', '是'],
                ['老年', '否', '是', '非常好', '是'],
                ['老年', '否', '是', '好', '是'],
                ['老年', '是', '否', '好', '是'],
                ['老年', '是', '否', '非常好', '是'],
                ['老年', '否', '否', '一般', '否']]
    labels = [u'年龄', u'有工作', u'有自己的房子', u'信贷情况', u'类别']
    # 返回数据集和每个维度的名称
    return datasets, labels

datasets, labels = create_data()

train_data = pd.DataFrame(datasets, columns=labels)
```

```
#print (train_data)

# 熵
def calc_ent(datasets):
    data_length = len(datasets)
    label_count = {}
    for i in range(data_length):
        label = datasets[i][-1]
        if label not in label_count:
            label_count[label] = 0
        label_count[label] += 1
    ent = -sum([(p/data_length)*log(p/data_length, 2) for p in label_count.values()])
    return ent

# 经验条件熵
def cond_ent(datasets, axis=0):
    data_length = len(datasets)
    feature_sets = {}
    for i in range(data_length):
        feature = datasets[i][axis]
        if feature not in feature_sets:
            feature_sets[feature] = []
        feature_sets[feature].append(datasets[i])
    cond_ent = sum([(len(p)/data_length)*calc_ent(p) for p in feature_sets.values()])
    return cond_ent
```

```python
# 信息增益
def info_gain(ent, cond_ent):
    return ent - cond_ent

def info_gain_train(datasets):
    count = len(datasets[0]) - 1
    ent = calc_ent(datasets)
    best_feature = []
    for c in range(count):
        c_info_gain = info_gain(ent, cond_ent(datasets, axis=c))
        best_feature.append((c, c_info_gain))
        print('特征({}) - info_gain - {:.3f}'.format(labels[c], c_info_gain))
    # 比较大小
    best_ = max(best_feature, key=lambda x: x[-1])
    return '特征({})的信息增益最大，选择为根节点特征'.format(labels[best_[0]])

info_gain_train(np.array(datasets))
```

```python
# 定义节点类 二叉树
class Node:
    def __init__(self, root=True, label=None, feature_name=None, feature=None):
        self.root = root
        self.label = label
        self.feature_name = feature_name
        self.feature = feature
        self.tree = {}
        self.result = {'label:': self.label, 'feature': self.feature, 'tree': self.tree}

    def __repr__(self):
        return '{}'.format(self.result)

    def add_node(self, val, node):
        self.tree[val] = node

    def predict(self, features):
        if self.root is True:
            return self.label
        return self.tree[features[self.feature]].predict(features)

class DTree:
    def __init__(self, epsilon=0.1):
        self.epsilon = epsilon
        self._tree = {}

    # 熵
    @staticmethod
    def calc_ent(datasets):
        data_length = len(datasets)
        label_count = {}
        for i in range(data_length):
            label = datasets[i][-1]
            if label not in label_count:
                label_count[label] = 0
            label_count[label] += 1
        ent = -sum([(p/data_length)*log(p/data_length, 2) for p in label_count.values()])
        return ent

    # 经验条件熵
    def cond_ent(self, datasets, axis=0):
        data_length = len(datasets)
        feature_sets = {}
        for i in range(data_length):
            feature = datasets[i][axis]
            if feature not in feature_sets:
                feature_sets[feature] = []
            feature_sets[feature].append(datasets[i])
        cond_ent = sum([(len(p)/data_length)*self.calc_ent(p) for p in feature_sets.values()])
        return cond_ent
```

```
    # 信息增益
    @staticmethod
    def info_gain(ent, cond_ent):
        return ent - cond_ent

    def info_gain_train(self, datasets):
        count = len(datasets[0]) - 1
        ent = self.calc_ent(datasets)
        best_feature = []
        for c in range(count):
            c_info_gain = self.info_gain(ent, self.cond_ent(datasets, axis=c))
            best_feature.append((c, c_info_gain))
        # 比较大小
        best_ = max(best_feature, key=lambda x: x[-1])
        return best_
```

```
def train(self, train_data):
    """
    input:数据集D(DataFrame格式)，特征集A，阈值eta
    output:决策树T """
    _, y_train, features = train_data.iloc[:, :-1], train_data.iloc[:, -1], train_data.columns[:-1]
    # 1,若D中实例属于同一类Ck，则T为单节点树，并将类Ck作为结点的类标记，返回T
    if len(y_train.value_counts()) == 1:
        return Node(root=True,
                    label=y_train.iloc[0])

    # 2, 若A为空，则T为单节点树，将D中实例树最大的类Ck作为该节点的类标记，返回T
    if len(features) == 0:
        return Node(root=True, label=y_train.value_counts().sort_values(ascending=False).index[0])

    # 3,计算最大信息增益 同5.1,Ag为信息增益最大的特征
    max_feature, max_info_gain = self.info_gain_train(np.array(train_data))
    max_feature_name = features[max_feature]
```

```
    # 4,Ag的信息增益小于阈值eta,则置T为单节点树，并将D中是实例数最大的类Ck作为该节点的类标记，返回T
    if max_info_gain < self.epsilon:
        return Node(root=True, label=y_train.value_counts().sort_values(ascending=False).index[0])

    # 5,构建Ag子集
    node_tree = Node(root=False, feature_name=max_feature_name, feature=max_feature)

    feature_list = train_data[max_feature_name].value_counts().index
    for f in feature_list:
        sub_train_df = train_data.loc[train_data[max_feature_name] == f].drop([max_feature_name], axis=1)

        # 6, 递归生成树
        sub_tree = self.train(sub_train_df)
        node_tree.add_node(f, sub_tree)

    # pprint.pprint(node_tree.tree)
    return node_tree
```

```
    def fit(self, train_data):
        self._tree = self.train(train_data)
        return self._tree

    def predict(self, X_test):
        return self._tree.predict(X_test)

datasets, labels = create_data()
data_df = pd.DataFrame(datasets, columns=labels)
dt = DTree()
tree = dt.fit(data_df)
print (tree)

print (dt.predict(['老年', '否', '否', '一般']))
```