朋友圈社交网络

列表List

当我们需要存储一组相似数据的时候，使用Python列表是一个很好的选择。列表是Python中最常用的数据类型之一，它允许我们在单个变量中存储多个值，并按照需要对这些值进行排序、添加、删除和修改。

在Python中，列表使用一对方括号[ ]来表示，其中各个元素之间用逗号分隔。列表中的每个元素都可以是任意数据类型，包括数字、字符串、布尔值、列表等等。

列表的定义

In [13]:

```
friends=["Alice","Bob","Cindy","Frank"] #定义字符列表
numbers=[1,3,6,5] #定义数值列表
```

列表元素的访问

一旦我们创建了一个列表，就可以使用索引来访问列表中的元素。在Python中，列表的索引从0开始，因此第一个元素的索引为0，第二个元素的索引为1，以此类推。例如，要访问上面创建的数字列表中的第一个元素，可以使用以下代码：

In [3]:

```
friends[0]
```

Out[3]:

'Alice'

In [4]:

```
friends[1]
```

Out[4]:

'Bob'

In [6]:

```
numbers[0]
```

Out[6]:

1

In [7]:

```
numbers[3]
```

Out[7]:

5

用列表元素参与运算

In [8]:

```
numbers[2]+numbers[3]
```

Out[8]:

11

In [9]:

```
friends[0]+friends[1]
```

Out[9]:

'AliceBob'

访问一个不存在的索引

In [10]:

```
friends[4]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_12844\3106893239.py in <module>
----> 1 friends[4]

IndexError: list index out of range
```

列表的方法（内置函数）

插入列表元素

In [14]:

```
friends.insert(3,"UU")
print(friends)
```

```
['Alice', 'Bob', 'Cindy', 'UU', 'Frank']
```

追加列表元素

In [15]:

```
friends.append("ZZ")
print(friends)
```

```
['Alice', 'Bob', 'Cindy', 'UU', 'Frank', 'ZZ']
```

统计列表长度

In [17]:

```
len(friends)
```

Out[17]:

5

删除列表元素

In [16]:

```
friends.remove("UU")
print(friends)
```

```
['Alice', 'Bob', 'Cindy', 'Frank', 'ZZ']
```

列表元素排序（数值列表）

In [21]:

```
numbers.sort()
print(numbers)
```

```
[1, 3, 5, 6]
```

小练习

题目描述： 编写一个程序，输入4个整数，并将它们保存到一个列表中。然后，在列表末尾追加一个比列表第0个元素大1的元素，最后，将列表中的元素按照从小到大的顺序进行排序，并输出排序后的列表。
示例:
输入: 1,3,4,2
输出: 1,2,2,3,4

In [26]:

```
#用户输入
a1=input("请输入第一个元素：")
a2=input("请输入第二个元素：")
a3=input("请输入第三个元素：")
a4=input("请输入第四个元素：")

#类型转换
a1=int(a1)
a2=int(a2)
a3=int(a3)
a4=int(a4)

#创建列表
L=[a1,a2,a3,a4]

#第0个元素+1
a5=a1+1

#该元素追加到末尾
L.append(a5)

#列表排序
L.sort()

#输出列表
print(L)
```

```
请输入第一个元素：8
请输入第二个元素：6
请输入第三个元素：7
请输入第四个元素：9
[6, 7, 8, 9, 9]
```

In [25]:

```
L
```

Out[25]:

['1', '2', '3', '4']

二维列表

In [27]:

```
#二维列表的创建
L=[[1,2,3],[4,5,6],[7,8,9]]
print(L)
```

[[1, 2, 3], [4, 5, 6], [7, 8, 9]]

In [29]:

```
#二维列表的索引
L[2][2]
```

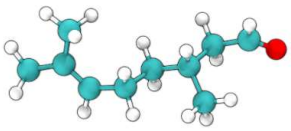
Out[29]:

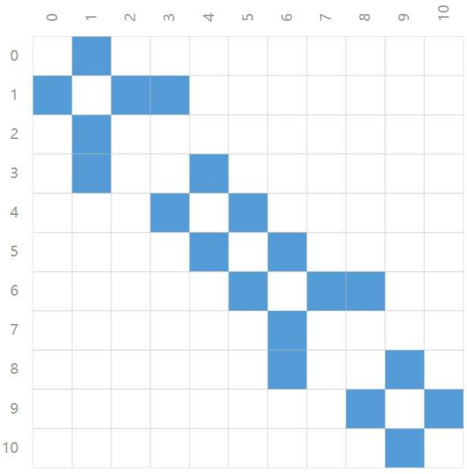
9

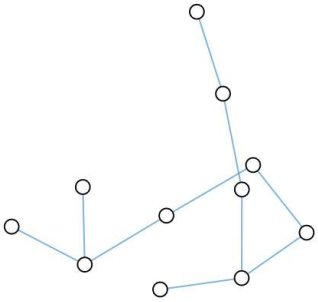
社交网络可视化

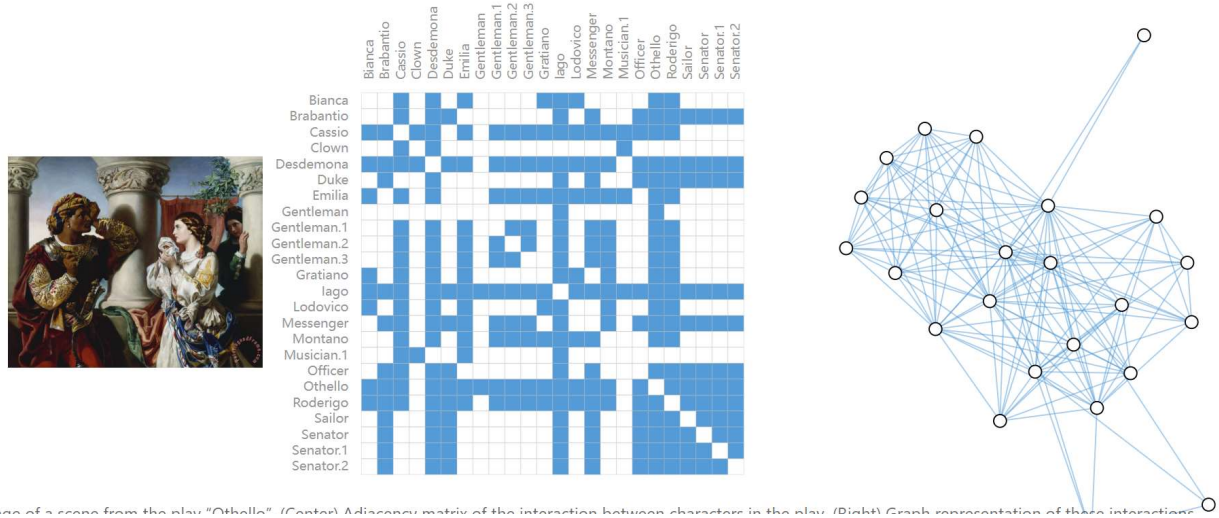
邻接矩阵

在社交网络可视化中，邻接矩阵是一种常见的表示方法。邻接矩阵是一个二维数组，用于表示图中各个节点之间的关系。如果两个节点之间有边相连，则在邻接矩阵中对应元素的值为1；否则为0。









(Left) Image of a scene from the play "Othello". (Center) Adjacency matrix of the interaction between characters in the play. (Right) Graph representation of these interactions.

In [31]:

```
L=[[0,0,1,0],[0,0,1,1],[1,1,0,1],[0,1,1,0]]
```

在上述的社会网络中，我们需要编写一个程序，输入两个用户的姓名编号，判断这两个用户之间是否认识。如果是，输出“好友关系存在”；否则，输出“好友关系不存在”。

In [44]:

```
L=[[0,1,1,0],[1,0,1,1],[1,1,0,1],[0,1,1,0]]

#输入姓名
name1=input("请输入第一个用户的姓名编号：")
name2=input("请输入第二个用户的姓名编号：")

#数值类型转换
name1=int(name1)
name2=int(name2)

if L[name1][name2]==0:
    print("好友关系不存在")
else:
    print("好友关系存在")
```

```
请输入第一个用户的姓名编号：0
请输入第二个用户的姓名编号：1
好友关系存在
```

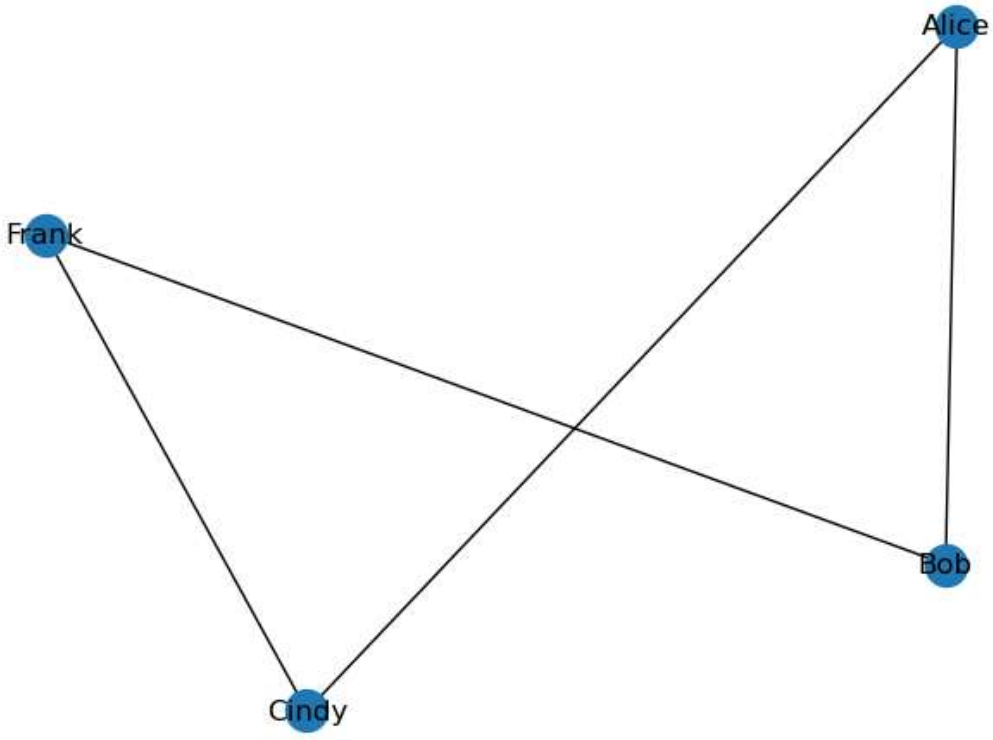
如何可视化?

In [38]:

```
#导入模块
import matplotlib.pyplot as plt
import networkx as nx

#创建一个空的网络（图）
G=nx.Graph()

#添加节点
G.add_nodes_from(["Alice","Bob","Cindy","Frank"])

#添加边
G.add_edge("Alice","Bob")
G.add_edge("Alice","Cindy")
G.add_edge("Frank","Cindy")
G.add_edge("Frank","Bob")
```

In [40]:

```
nx.draw(G,with_labels=True)
plt.show()
```



课堂练习： 在上述的社交网络中，如何你想发朋友圈屏蔽Alice，你还应该屏蔽哪些人？请写一个程序完成该任务
示例： 输入：0
输出：1,2

In [52]:

```
#用户输入
name=input("请输入你的目标屏幕对象：")

#数据类型转换
name=int(name)

if  name in [0,1,2,3]:

    #查找她/他认识的人

    if L[name][0]==1: #如果目标对象跟第0号对象认识
        print(0)
    if L[name][1]==1: #如果目标对象跟第1号对象认识
        print(1)
    if L[name][2]==1: #如果目标对象跟第2号对象认识
        print(2)
    if L[name][3]==1: #如果目标对象跟第3号对象认识
        print(3)
else:
    print("输入错误！")

'''
for i in range(3):
    if L[name][i]==1:
        print(i)
'''
```

```
请输入你的目标屏幕对象：3
1
2
```

Out[52]:

```
'\nfor i in range(3):\n    if L[name][i]==1:\n        print(i)\n'
```

In [43]:

```
L[0][1]
```

Out[43]:

```
0
```

进阶练习：只根据网络层级标准来判断应该屏蔽的人是否完全合理（无风险？） 是否可以考虑按照人与人之间连接的紧密程度来判断应该屏蔽哪些人？

In [ ]:

```
```