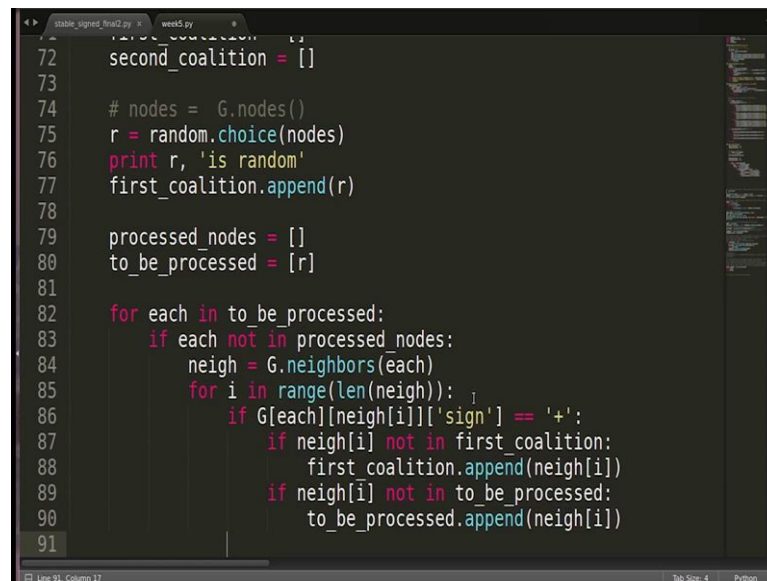


**Social Networks**  
**Prof. S. R. S. Iyengar**  
**Department of Computer Science**  
**Indian Institute of Technology, Ropar**

**Homophily (Continued) & Positive and Negative Relationships**  
**Lecture – 73**  
**Forming two coalitions (Continued)**

(Refer Slide Time: 00:05)



```
72 second_coalition = []
73
74 # nodes = G.nodes()
75 r = random.choice(nodes)
76 print r, 'is random'
77 first_coalition.append(r)
78
79 processed_nodes = []
80 to_be_processed = [r]
81
82 for each in to_be_processed:
83     if each not in processed_nodes:
84         neigh = G.neighbors(each)
85         for i in range(len(neigh)):
86             if G[each][neigh[i]]['sign'] == '+':
87                 if neigh[i] not in first_coalition:
88                     first_coalition.append(neigh[i])
89                 if neigh[i] not in to_be_processed:
90                     to_be_processed.append(neigh[i])
91
```

We will take nodes from this list to be processed and we will keep checking their neighbors. So, we will start a loop for each in to be processed, if each not in process node. So, if this node has not already been processed, if each not in processed node then we will look at his neighbors; so let us have this list `neigh[i]`s equal to `G dot neighbors` we have to look at the neighbors of each. So, we will write like this.

Now we have take these neighbors one by one and we have to see whether these neighbors are friends or enemies. If they are friends, they will be added to the first list, if they are enemies they will be added to the second list. Apart from that if they are friends and they are being added to the first list, they have to be processed afterwards. So, they will be added to the list to be processed.

And if they are enemies and they are being added to the second list, then they will not be processed further. So, they will be added to the list process nodes. So, we will write for `i` in range, length of this list because to get to know how many neighbors are there of

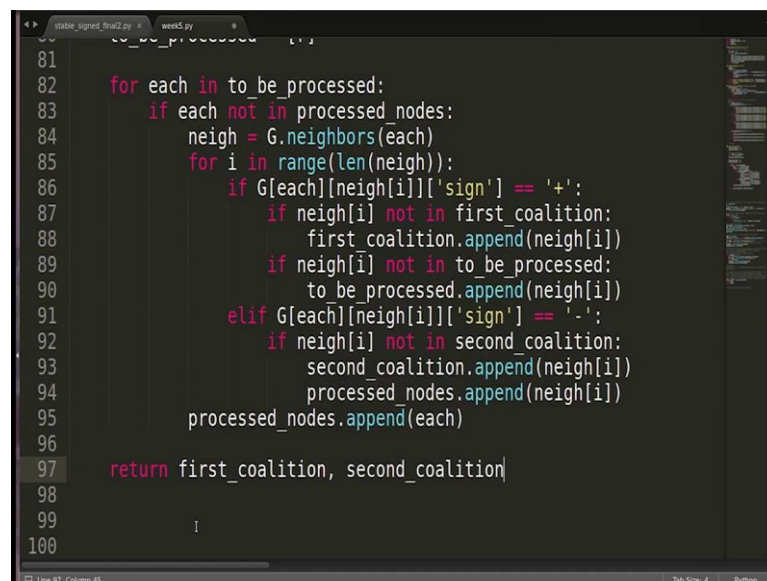
course, in this particular example there are  $n - 1$  precisely  $n - 1$  neighbors, but we can generalize this and we can write up to length of neighbors.

So, once we take the neighbor, we have to check the relationship of this neighbor with the node that is each. So, we will write `G[each][neigh[i]]` and how do we check the relation by using this attribute `sign`. So, we will start this if this is equal to plus which means it is a friend, as I told you what has to be done it has to be added to the list of first coalition.

So, if it is not already there. So, we will check if `neigh[i]` not in first coalition, then it will be added there. So, you write `first_coalition.append(neigh[i])` done. And second thing that has to be done is it has to be added to the list of to be process nodes if it is not all be there. You will write if `neigh[i]` not in to be processed and it will appended there to be processed dot append `neigh[i]` alright

And in the other case when the neighbor is an enemy, what we are going to do is.

(Refer Slide Time: 02:55)



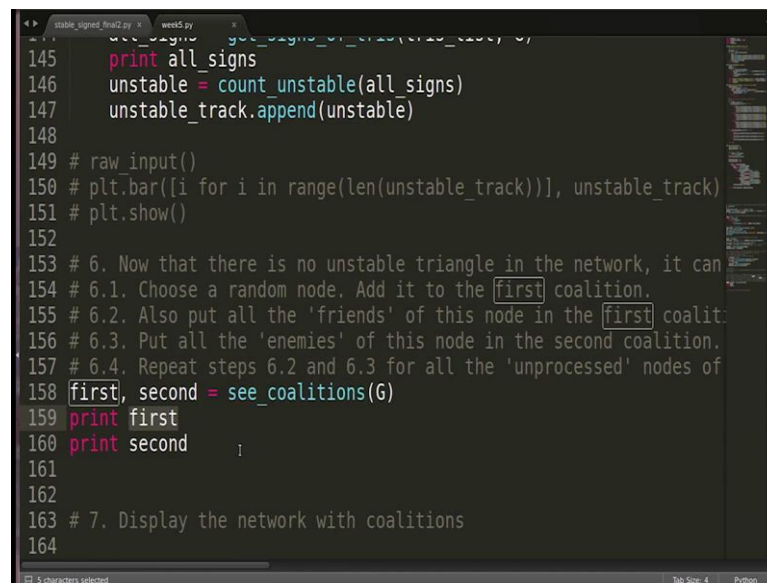
```
81
82     for each in to_be_processed:
83         if each not in processed_nodes:
84             neigh = G.neighbors(each)
85             for i in range(len(neigh)):
86                 if G[each][neigh[i]]['sign'] == '+':
87                     if neigh[i] not in first_coalition:
88                         first_coalition.append(neigh[i])
89                     if neigh[i] not in to_be_processed:
90                         to_be_processed.append(neigh[i])
91                 elif G[each][neigh[i]]['sign'] == '-':
92                     if neigh[i] not in second_coalition:
93                         second_coalition.append(neigh[i])
94                     processed_nodes.append(neigh[i])
95             processed_nodes.append(each)
96
97     return first_coalition, second_coalition
98
99
100
```

So, will write `elif` and this thing which means this is minus, so its an enemy. What has to be done? It has to be added to the second coalition list if it is not already there. So, we will write if `neigh[i]` not in `second_coalition`, it will be appended `neigh[i]` ok. Second in that has to be done is as I told you it does not have to be processed again. So, it has to be added to the list of processed nodes.

So, we can directly just add it there, `processed_nodes.append(neigh[i])`. Next thing that has to be done is this each the node which is called each has been process now. So, it has to be added to the list of process nodes. So, we will write here `processed_nodes.append` each ok. So, that is done

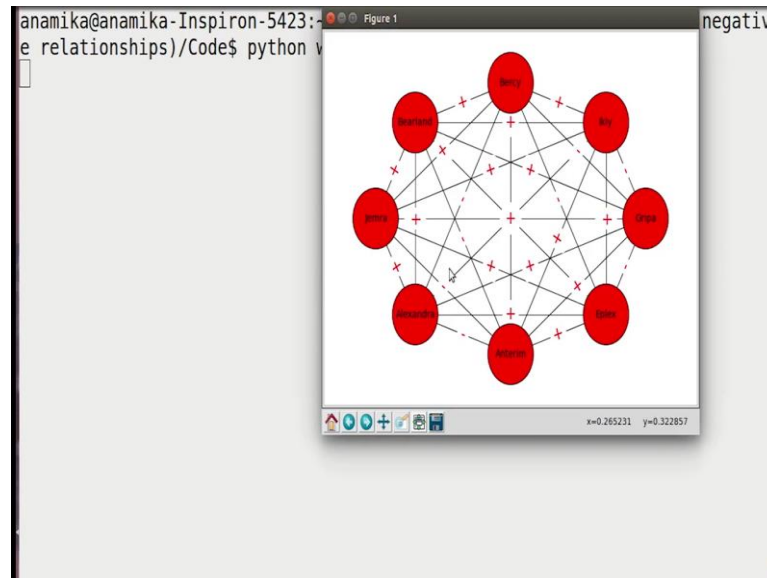
So, there this loop should basically divide the nodes in the two lists that is first coalition and second coalition and that is what this function will return. So, we are going to return this. So, we will write `return first_coalition, second_coalition` this should work.

(Refer Slide Time: 04:25)

A screenshot of a Python code editor with a dark theme. The code is written in a file named 'stable\_signed\_final2.py'. The visible lines of code are: 145: `print all_signs`, 146: `unstable = count_unstable(all_signs)`, 147: `unstable_track.append(unstable)`, 148: (empty line), 149: `# raw_input()`, 150: `# plt.bar([i for i in range(len(unstable_track))], unstable_track)`, 151: `# plt.show()`, 152: (empty line), 153: `# 6. Now that there is no unstable triangle in the network, it can`, 154: `# 6.1. Choose a random node. Add it to the first coalition.`, 155: `# 6.2. Also put all the 'friends' of this node in the first coalition.`, 156: `# 6.3. Put all the 'enemies' of this node in the second coalition.`, 157: `# 6.4. Repeat steps 6.2 and 6.3 for all the 'unprocessed' nodes of`, 158: `first, second = see_coalitions(G)`, 159: `print first`, 160: `print second`, 161: (empty line), 162: (empty line), 163: `# 7. Display the network with coalitions`, 164: (empty line). The code uses comments to describe the steps of an algorithm for forming coalitions in a network. The editor interface includes a file explorer on the right and a status bar at the bottom showing '5 characters selected' and 'Tab Size: 4 Python'.

Let us go down and see this is where we call this function and after it returned the two lists, we are just spending those two lists. So, let us see how it works ok.

(Refer Slide Time: 04:32)



These are the air cities and you see the relationships amongst them as well positive and negative relationships I am closing it.

(Refer Slide Time: 04:41)

[illegible]

So, it has processed, and you see these are the two coming the two coalitions formed in the first one there are 5 countries and in the second one there are 3 countries. So, this is the division

In the next video we are going to visualize this network and we are going to compare it with the initial configuration that was entered initially.