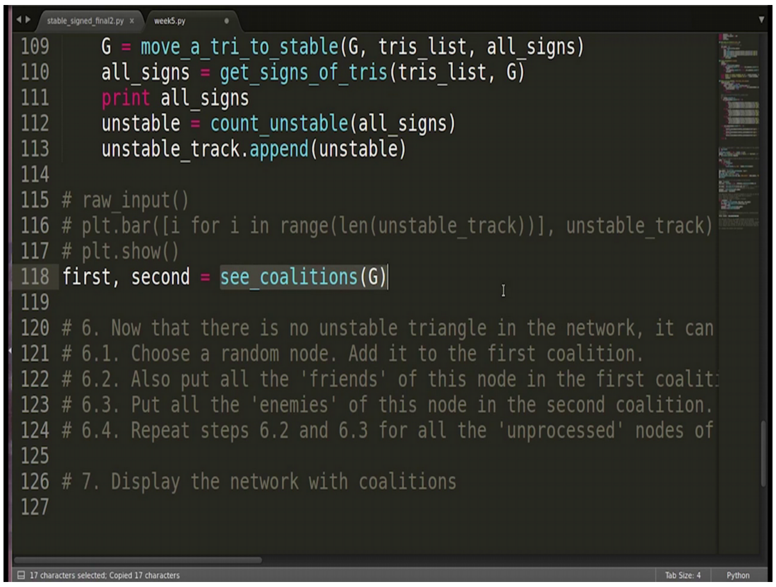
Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Homophily (Continued) & Positive and Negative Relationships
Lecture – 72
Forming two coalitions

(Refer Slide Time: 00:05)

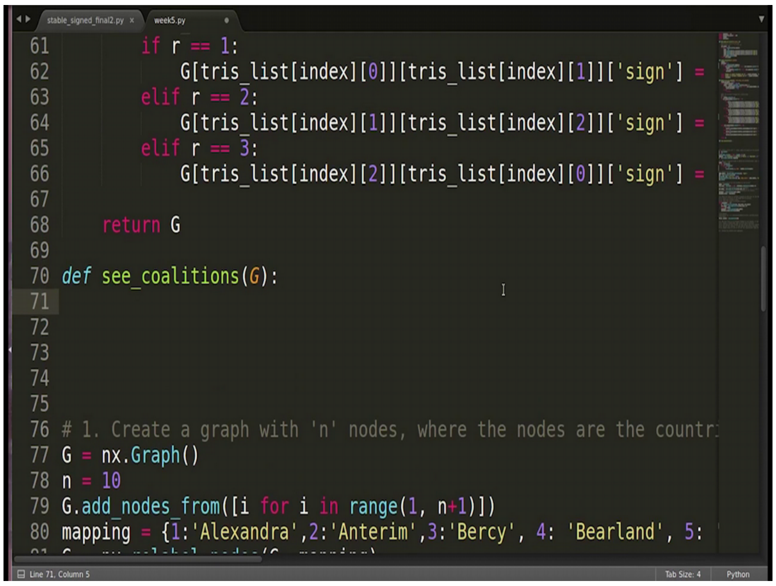


```
    G = move_a_tri_to_stable(G, tris_list, all_signs)
    all_signs = get_signs_of_tris(tris_list, G)
    print all_signs
    unstable = count_unstable(all_signs)
    unstable_track.append(unstable)

# raw_input()
# plt.bar([i for i in range(len(unstable_track))], unstable_track)
# plt.show()
first, second = see_coalitions(G)

# 6. Now that there is no unstable triangle in the network, it can
# 6.1. Choose a random node. Add it to the first coalition.
# 6.2. Also put all the 'friends' of this node in the first coalit
# 6.3. Put all the 'enemies' of this node in the second coalition.
# 6.4. Repeat steps 6.2 and 6.3 for all the 'unprocessed' nodes of

# 7. Display the network with coalitions
```

In the previous video we moved the network from an unstable state to a stable state, where there are no unstable triangles. So, once that happens it is possible to divide the nodes of the network into two groups. So, that is what we are going to do here now.

Let us create a function for that and let the function return to lists, where the first list will contain the nodes of the first row and second list will contain the nodes of the second row. So, let us create this function and let us call it see coalitions and let us pass the graph here.

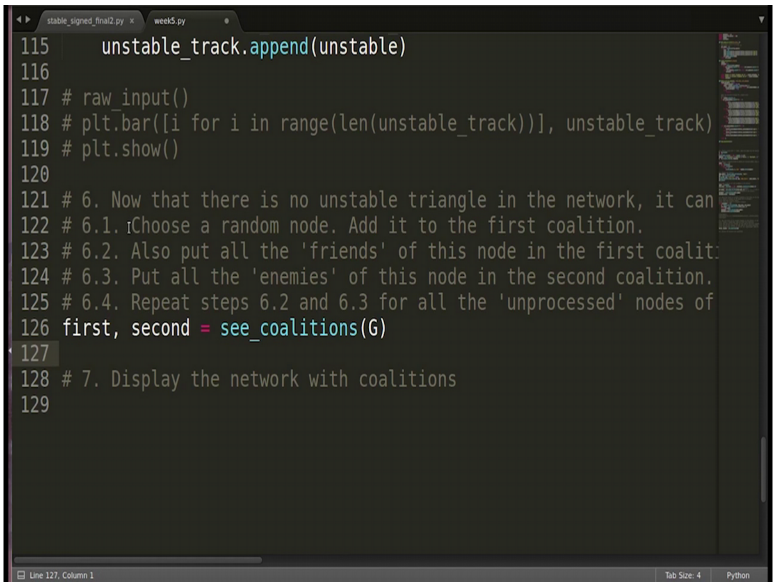
(Refer Slide Time: 00:53)



```
            if r == 1:
                G[tris_list[index][0]][tris_list[index][1]]['sign'] =
            elif r == 2:
                G[tris_list[index][1]][tris_list[index][2]]['sign'] =
            elif r == 3:
                G[tris_list[index][2]][tris_list[index][0]]['sign'] =

    return G

def see_coalitions(G):





# 1. Create a graph with 'n' nodes, where the nodes are the countr
G = nx.Graph()
n = 10
G.add_nodes_from([i for i in range(1, n+1)])
mapping = {1:'Alexandra',2:'Anterim',3:'Bercy', 4: 'Bearland', 5:
```

So, let us create this function here let us go up and we will create this function. So, what do we have to do here? As I told you previously the strategy that we are going to follow as we explained here with me.

(Refer Slide Time: 01:04)

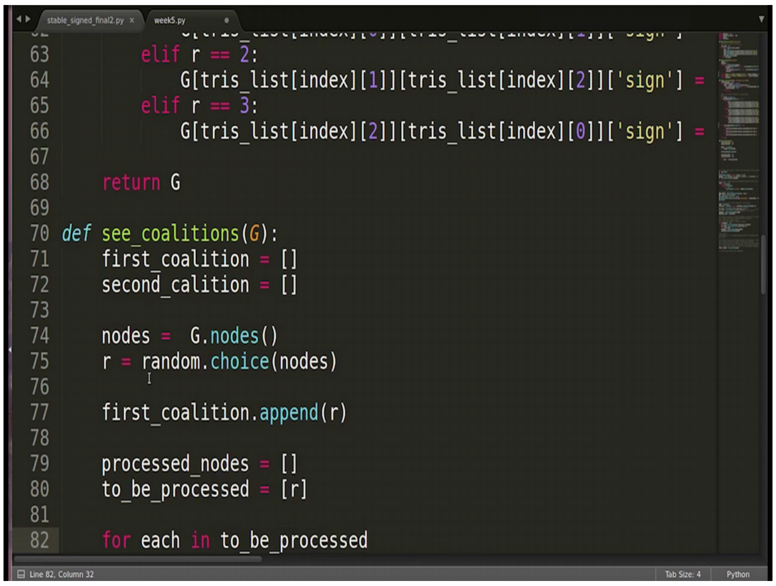


```
    unstable_track.append(unstable)

# raw_input()
# plt.bar([i for i in range(len(unstable_track))], unstable_track)
# plt.show()

# 6. Now that there is no unstable triangle in the network, it can
# 6.1. Choose a random node. Add it to the first coalition.
# 6.2. Also put all the 'friends' of this node in the first coalit
# 6.3. Put all the 'enemies' of this node in the second coalition.
# 6.4. Repeat steps 6.2 and 6.3 for all the 'unprocessed' nodes of
first, second = see_coalitions(G)

# 7. Display the network with coalitions

```

Let me write it here because this is the sixth step the second last step.

So, we are going to choose a random node and we will add it to the first coalition.

(Refer Slide Time: 01:18)



```
        elif r == 2:
            G[tris_list[index][1]][tris_list[index][2]]['sign'] =
        elif r == 3:
            G[tris_list[index][2]][tris_list[index][0]]['sign'] =

    return G

def see_coalitions(G):
    first_coalition = []
    second_calition = []

    nodes =  G.nodes()
    r = random.choice(nodes)

    first_coalition.append(r)

    processed_nodes = []
    to_be_processed = [r]

    for each in to_be_processed
```

Now, how do we choose a random node? Very simple, we will first access the list of nodes; nodes is equal to G dot nodes. So, we got a list of nodes now what we have to do is we have to choose one node randomly out of it. The function that we can use for that is ran int that we have previously used as well or we can simply use the function random dot choice which gives us one entry out of a given list. So, we have this list as nodes. So, we can use any of the functions.

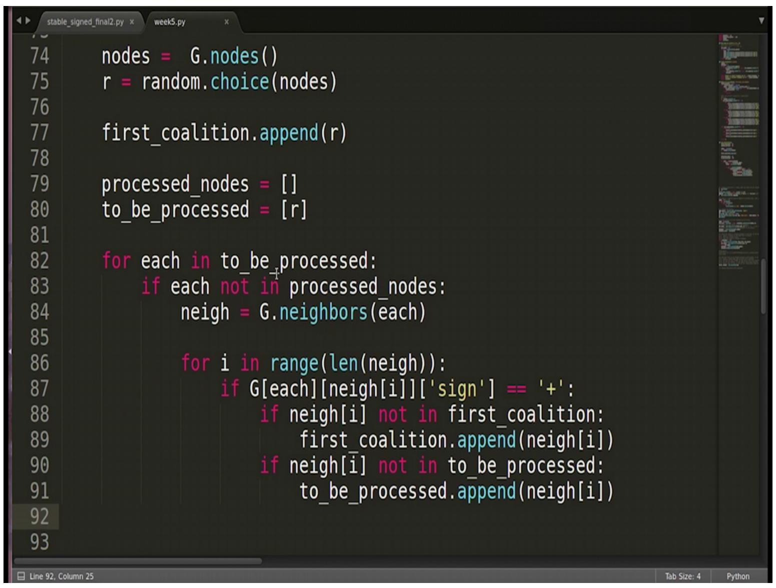
Let us use the function choice r is the random node. So, we will write random dot choice; the choice should be out of this list nodes. So, we got a random node here now since we have return also and we have to keep track also let us create two lists for each coalition, first coalition empty list and similarly second coalition another entry list. So, we are going to add nodes to this coalition.

Now, as I told you we will we will choose a random node and we will add it to the first coalition. So, we will write first I am sorry first coalition dot append r right now basically what we have to do is; we have to run a sort of a (Refer Time: 02:49) first search on this node we can added condition that all the enemies of this node are not going to be a part of the bfs tree right.

So, as I told you we will look at the neighbours of this node and some of these neighbours will be positive some of these neighbours will be its friends and some of its neighbours will be its enemies. So, the neighbours which are its friends are going to be a part of the bfs tree, they will be explored further. However, the nodes which are going to be which are its enemies are not going to be part of the bfs tree they will not be explored they will simply be put in the second coalition list. So, that is the strategy you can use any other strategy as well. So, this is one simple method that works well. So, as we do in bfs also we keep a track of the nodes which have been processed already the nodes which are yet to be processed. So, we are going in the same fashion.

So, let us create for that purpose two lists processed nodes initialized to empty to be processed same. So, this list will keep track of all the nodes which are to be processed. Now since r is the node which has to be processed we will put r into it ok. Now we will we have to keep running the loop of bfs until this to be processed list becomes empty right you will write for each in to be processed.

(Refer Slide Time: 04:32)



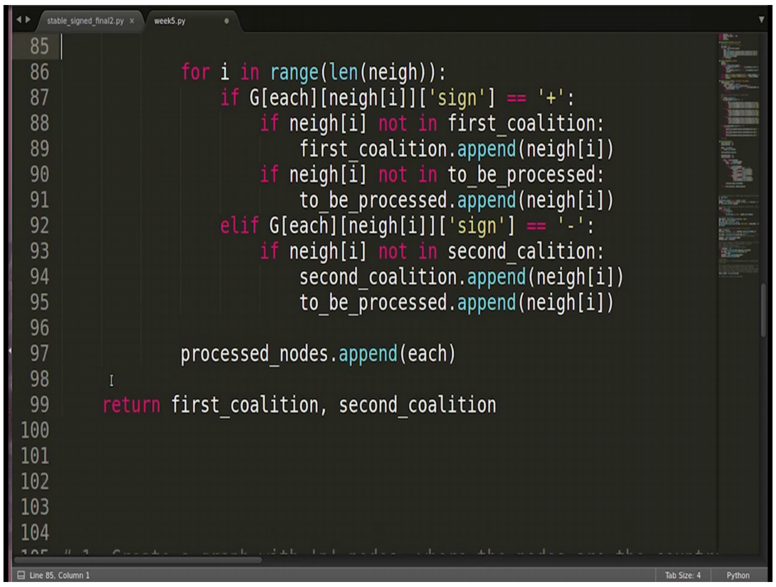
```
nodes =  G.nodes()
r = random.choice(nodes)

first_coalition.append(r)

processed_nodes = []
to_be_processed = [r]

for each in to_be_processed:
    if each not in processed_nodes:
        neigh = G.neighbors(each)

        for i in range(len(neigh)):
            if G[each][neigh[i]]['sign'] == '+':
                if neigh[i] not in first_coalition:
                    first_coalition.append(neigh[i])
                if neigh[i] not in to_be_processed:
                    to_be_processed.append(neigh[i])
```

Now, in case the node is an enemy what has to be done? It has to be added to the second coalition and it does not have to be processed. So, we will write elif let me copy paste if its a negative sign, we will add it to second coalition if it is not already there. So, we will check if neigh I not in second coalition we will add it there. So, we will write second coalition dot append neigh i that is one thing. Secondly, it does not have to be process. So, we are going to add it to be processed list, we will write to be processed dot append neigh i right, ok.

Now, by this time r node that is each which we started with has been completely processed. So, we will write here, we will write since it does that it has been processed we will write processed nodes dot append each. So, whatever we just processed, we will added to this list. So, that we have done.

So, this was the method to divide into first coalition and second coalition now this function has to return these two lists.

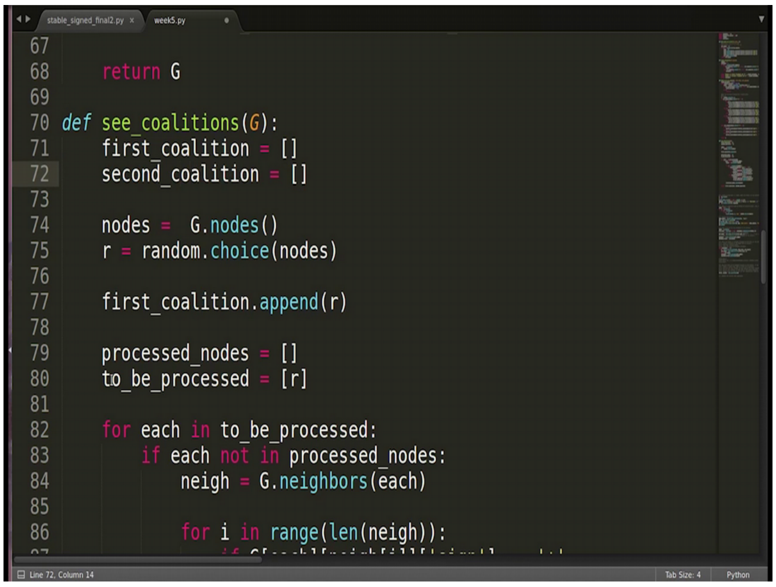
(Refer Slide Time: 05:57)



```
            for i in range(len(neigh)):
                if G[each][neigh[i]]['sign'] == '+':
                    if neigh[i] not in first_coalition:
                        first_coalition.append(neigh[i])
                    if neigh[i] not in to_be_processed:
                        to_be_processed.append(neigh[i])
                elif G[each][neigh[i]]['sign'] == '-':
                    if neigh[i] not in second_calition:
                        second_coalition.append(neigh[i])
                        to_be_processed.append(neigh[i])

            processed_nodes.append(each)

    return first_coalition, second_coalition
```

So, we will make it return here ok. We will write return first coalition second coalition if the spelling is wrong is it; oh I am sorry second coalition its wrong everywhere I am sorry where else here ok.

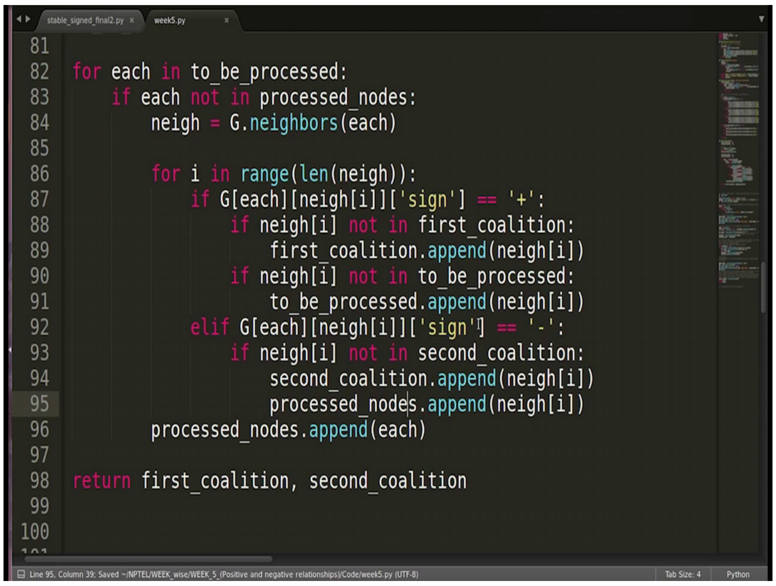
(Refer Slide Time: 06:23)



```
    return G

def see_coalitions(G):
    first_coalition = []
    second_coalition = []

    nodes =  G.nodes()
    r = random.choice(nodes)

    first_coalition.append(r)

    processed_nodes = []
    to_be_processed = [r]

    for each in to_be_processed:
        if each not in processed_nodes:
            neigh = G.neighbors(each)

            for i in range(len(neigh)):
```

I think we are done (Refer Time: 06:28).

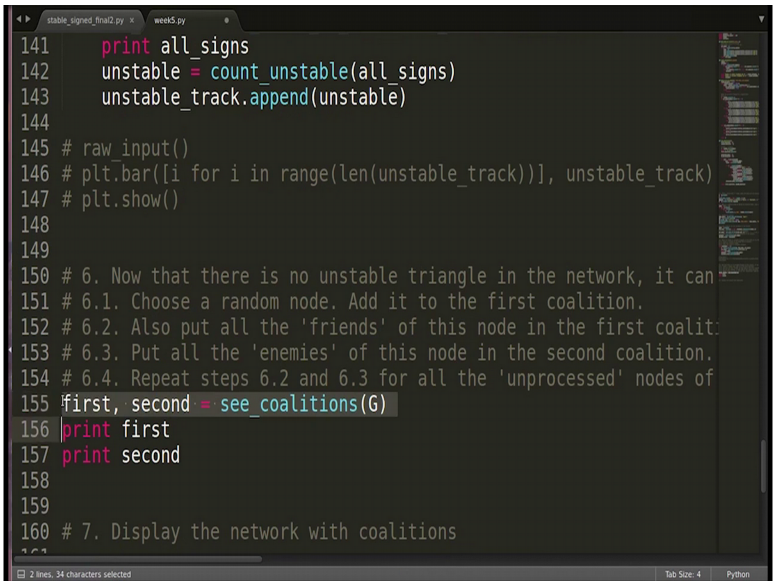
(Refer Slide Time: 06:28)


stable_signed_final2.py
week5.py
Line 95, Column 39; Saved ~/NPTEL/WEEK_wise/WEEK_5_(Positive and negative relationships)/Code/week5.py (UTF-8)
Tab Size: 4
Python

```
    for each in to_be_processed:
        if each not in processed_nodes:
            neigh = G.neighbors(each)

            for i in range(len(neigh)):
                if G[each][neigh[i]]['sign'] == '+':
                    if neigh[i] not in first_coalition:
                        first_coalition.append(neigh[i])
                    if neigh[i] not in to_be_processed:
                        to_be_processed.append(neigh[i])
                elif G[each][neigh[i]]['sign'] == '-':
                    if neigh[i] not in second_coalition:
                        second_coalition.append(neigh[i])
                        processed_nodes.append(neigh[i])
            processed_nodes.append(each)

    return first_coalition, second_coalition
```

We have added the friends of a given node to the first coalition and the enemies of the given node to the second coalition, and we are not processing the nodes which are going to go in the second coalition. Here, I am really sorry we do not need to process the nodes which are which are added to the second coalition. So, they will go to processed nodes here, because they are not going to be processed again ok.

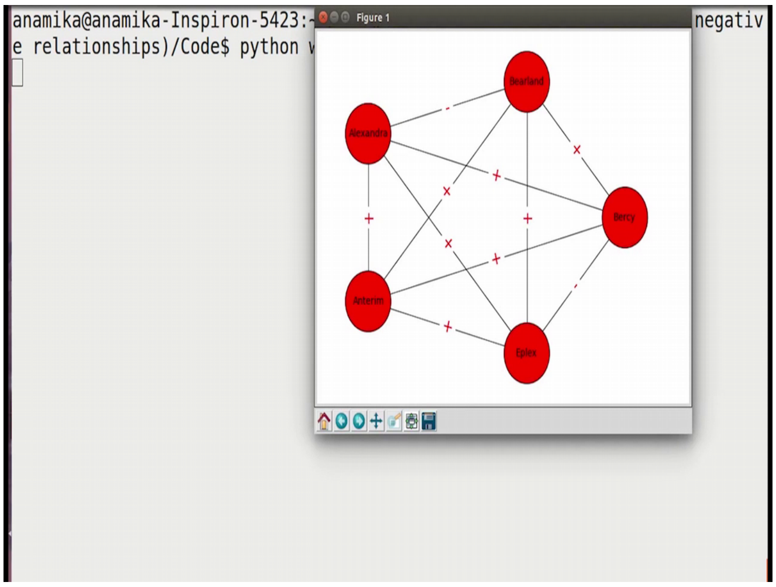
(Refer Slide Time: 07:00)



```
    print all_signs
    unstable = count_unstable(all_signs)
    unstable_track.append(unstable)

# raw_input()
# plt.bar([i for i in range(len(unstable_track))], unstable_track)
# plt.show()


# 6. Now that there is no unstable triangle in the network, it can
# 6.1. Choose a random node. Add it to the first coalition.
# 6.2. Also put all the 'friends' of this node in the first coalit
# 6.3. Put all the 'enemies' of this node in the second coalition.
# 6.4. Repeat steps 6.2 and 6.3 for all the 'unprocessed' nodes of
first, second = see_coalitions(G)
print first
print second


# 7. Display the network with coalitions
```

So, this is the function and let us go back here and see if it works well. So, we call this function see coalitions and its returning the two lists of nodes first and second and then we are printing these two lists.

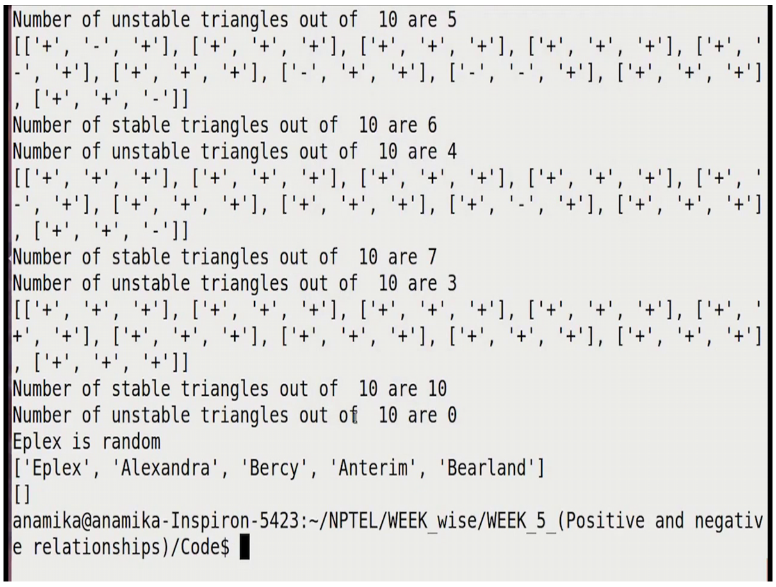
(Refer Slide Time: 07:15)



```
anamika@anamika-Inspiron-5423:-                    negativ
e relationships)/Code$ python v
```

Let us see how it functions. So, let us run this, this is the nodes this is the networks with five countries I am closing it. So, that it can go ahead.

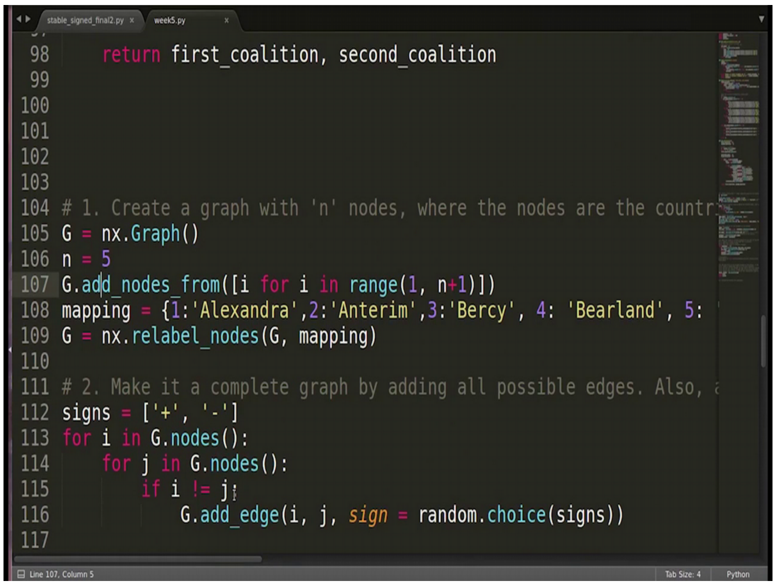
(Refer Slide Time: 07:30)



```
Number of unstable triangles out of  10 are 5
[['+', '-', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '
-', '+'], ['+', '+', '+'], ['-', '+', '+'], ['-', '-', '+'], ['+', '+', '+']
, ['+', '+', '-']]
Number of stable triangles out of  10 are 6
Number of unstable triangles out of  10 are 4
[['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '
-', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '-', '+'], ['+', '+', '+']
, ['+', '+', '-']]
Number of stable triangles out of  10 are 7
Number of unstable triangles out of  10 are 3
[['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '
+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+'], ['+', '+', '+']
, ['+', '+', '+']]
Number of stable triangles out of  10 are 10
Number of unstable triangles out of  10 are 0
Eplex is random
['Eplex', 'Alexandra', 'Bercy', 'Anterim', 'Bearland']
[]
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_5_(Positive and negativ
e relationships)/Code$ 
```

Let me show you one some more countries.

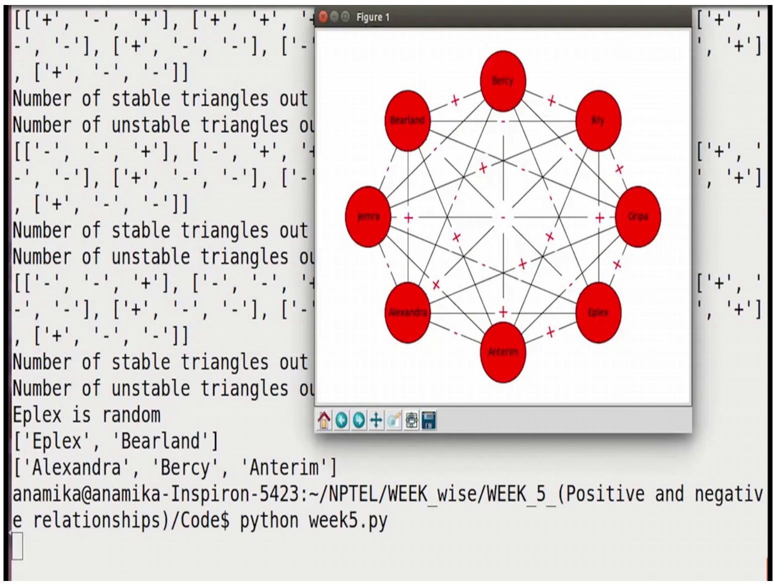
(Refer Slide Time: 07:34)



```python
    return first_coalition, second_coalition




# 1. Create a graph with 'n' nodes, where the nodes are the countr
G = nx.Graph()
n = 5
G.add_nodes_from([i for i in range(1, n+1)])
mapping = {1:'Alexandra',2:'Anterim',3:'Bercy', 4: 'Bearland', 5: 
G = nx.relabel_nodes(G, mapping)

# 2. Make it a complete graph by adding all possible edges. Also, a
signs = ['+', '-']
for i in G.nodes():
    for j in G.nodes():
        if i != j:
            G.add_edge(i, j, sign = random.choice(signs))

```

So, that we are able to nicely see the division into two groups: let me increase this to 8 and in c.

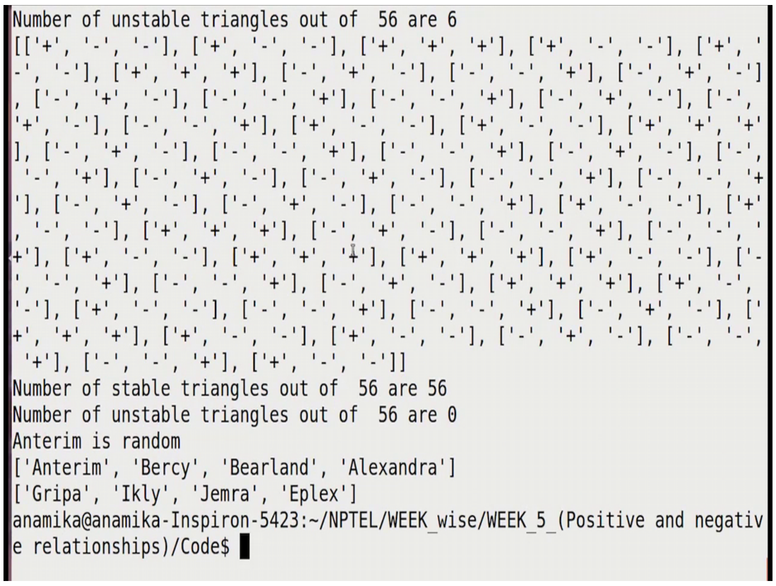
(Refer Slide Time: 07:44)



```
[['+', '-', '+'], ['+', '+', '                                        ['+', '
-', '-'], ['+', '-', '-'], ['-                                        ', '+']
, ['+', '-', '-']]
Number of stable triangles out
Number of unstable triangles ou
[['-', '-', '+'], ['-', '+', '                                        ['+', '
-', '-'], ['+', '-', '-'], ['-                                        ', '+']
, ['+', '-', '-']]
Number of stable triangles out
Number of unstable triangles ou
[['-', '-', '+'], ['-', '-', '                                        ['+', '
-', '-'], ['+', '-', '-'], ['-                                        ', '+']
, ['+', '-', '-']]
Number of stable triangles out
Number of unstable triangles ou
Eplex is random
['Eplex', 'Bearland']
['Alexandra', 'Bercy', 'Anterim']
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_5_(Positive and negativ
e relationships)/Code$ python week5.py
```

So, these are the 8 countries let me close this.

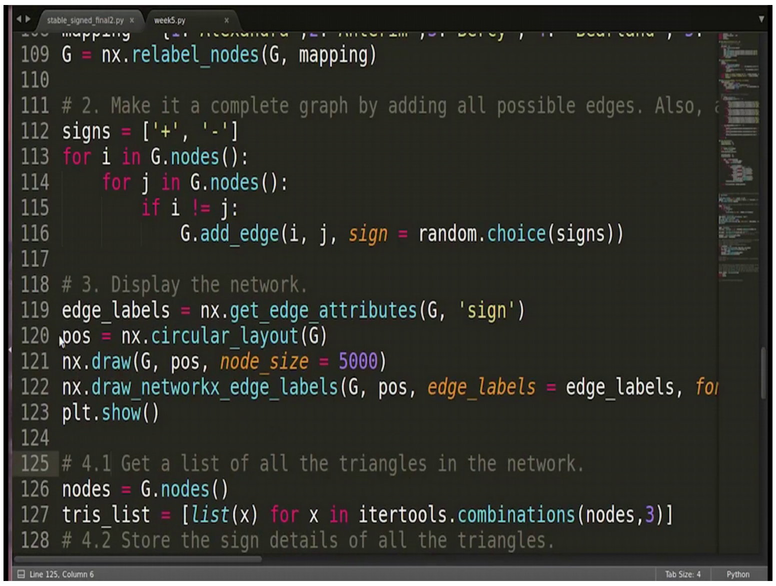
(Refer Slide Time: 07:46)



```
Number of unstable triangles out of  56 are 6
[['+', '-', '-'], ['+', '-', '-'], ['+', '+', '+'], ['+', '-', '-'], ['+', '
-', '-'], ['+', '+', '+'], ['-', '+', '-'], ['-', '-', '+'], ['-', '+', '-']
, ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '+'], ['-', '+', '-'], ['-',
'+', '-'], ['-', '-', '+'], ['+', '-', '-'], ['+', '-', '-'], ['+', '+', '+'
], ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '+'], ['-', '+', '-'], ['-',
 '-', '+'], ['-', '+', '-'], ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '+
'], ['-', '+', '-'], ['-', '+', '-'], ['-', '-', '+'], ['+', '-', '-'], ['+'
, '-', '-'], ['+', '+', '+'], ['-', '+', '-'], ['-', '-', '+'], ['-', '-', '
+'], ['+', '-', '-'], ['+', '+', '+'], ['+', '+', '+'], ['+', '-', '-'], ['-
', '-', '+'], ['-', '-', '+'], ['-', '+', '-'], ['+', '+', '+'], ['+', '-',
'-'], ['+', '-', '-'], ['-', '-', '+'], ['-', '-', '+'], ['-', '+', '-'], ['
+', '+', '+'], ['+', '-', '-'], ['+', '-', '-'], ['-', '+', '-'], ['-', '-',
 '+'], ['-', '-', '+'], ['+', '-', '-']]
Number of stable triangles out of  56 are 56
Number of unstable triangles out of  56 are 0
Anterim is random
['Anterim', 'Bercy', 'Bearland', 'Alexandra']
['Gripa', 'Ikly', 'Jemra', 'Eplex']
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_5_(Positive and negativ
e relationships)/Code$
```

So, it move the triangle to stable state and then it divided into two groups. As you can see out of the 8 countries four are in first group and the rest four are in the second group if we want to visualize as we can see.

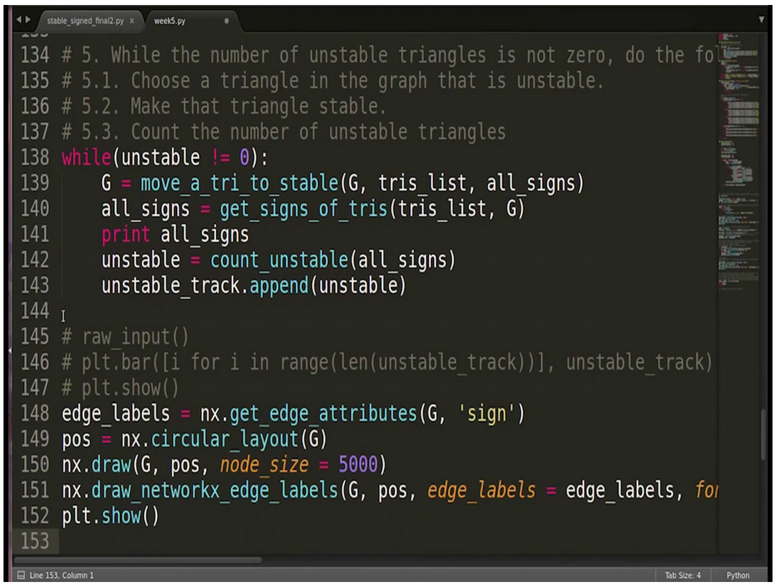
(Refer Slide Time: 08:08)



```
G = nx.relabel_nodes(G, mapping)

# 2. Make it a complete graph by adding all possible edges. Also, a
signs = ['+', '-']
for i in G.nodes():
    for j in G.nodes():
        if i != j:
            G.add_edge(i, j, sign = random.choice(signs))

# 3. Display the network.
edge_labels = nx.get_edge_attributes(G, 'sign')
pos = nx.circular_layout(G)
nx.draw(G, pos, node_size = 5000)
nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labels, fo
plt.show()

# 4.1 Get a list of all the triangles in the network.
nodes = G.nodes()
tris_list = [list(x) for x in itertools.combinations(nodes,3)]
# 4.2 Store the sign details of all the triangles.
```

Let me add code of displaying the communities as well the coalitions ok. This is how we displayed the graph previously; I am just copying the code; so that we can display it once again.

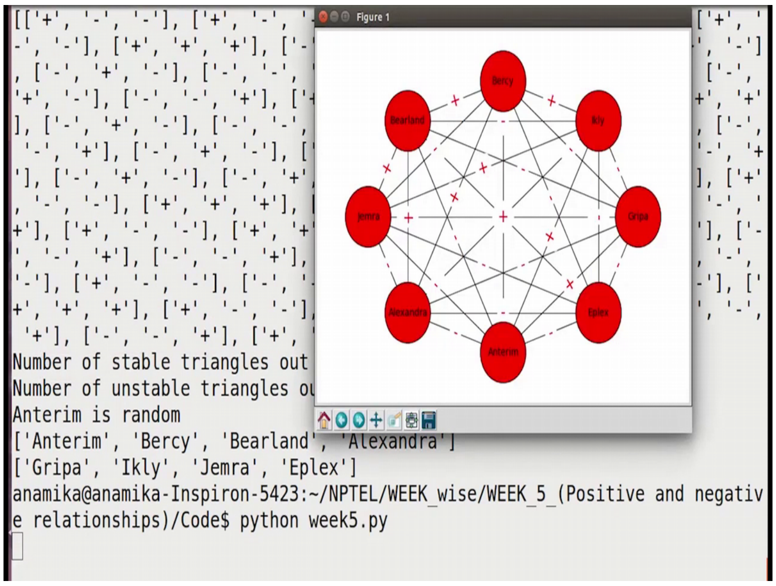
(Refer Slide Time: 08:21)



```
# 5. While the number of unstable triangles is not zero, do the fo
# 5.1. Choose a triangle in the graph that is unstable.
# 5.2. Make that triangle stable.
# 5.3. Count the number of unstable triangles
while(unstable != 0):
    G = move_a_tri_to_stable(G, tris_list, all_signs)
    all_signs = get_signs_of_tris(tris_list, G)
    print all_signs
    unstable = count_unstable(all_signs)
    unstable_track.append(unstable)

# raw_input()
# plt.bar([i for i in range(len(unstable_track))], unstable_track)
# plt.show()
edge_labels = nx.get_edge_attributes(G, 'sign')
pos = nx.circular_layout(G)
nx.draw(G, pos, node_size = 5000)
nx.draw_networkx_edge_labels(G, pos, edge_labels = edge_labels, fo
plt.show()
```

So, I am executing it again.

(Refer Slide Time: 08:24)



```
[['+', '-', '-'], ['+', '-', '                                 ['+', '
-', '-'], ['+', '+', '+'], ['-                                 ', '-']
, ['-', '+', '-'], ['-', '-',                                  ['-',
'+', '-'], ['-', '-', '+'], ['+                                +', '+'
], ['-', '+', '-'], ['-', '-',                                 , ['-',
 '-', '+'], ['-', '+', '-'], [                                 '-', '+
'], ['-', '+', '-'], ['-', '+',                                ], ['+'
, '-', '-'], ['+', '+', '+'],                                  '-', '
+'], ['+', '-', '-'], ['+', '+                                 '], ['-
', '-', '+'], ['-', '-', '+'],                                 , '-',
'-'], ['+', '-', '-'], ['-', '                                 -'], ['
+', '+', '+'], ['+', '-', '-'],                                ', '-',
 '+'], ['-', '-', '+'], ['+',
Number of stable triangles out
Number of unstable triangles ou
Anterim is random
['Anterim', 'Bercy', 'Bearland', 'Alexandra']
['Gripa', 'Ikly', 'Jemra', 'Eplex']
anamika@anamika-Inspiron-5423:~/NPTEL/WEEK_wise/WEEK_5_(Positive and negativ
e relationships)/Code$ python week5.py
```

So, this is the initial network; let us take one example triangle. So, Bearland Jenera and Bercy they have an unstable relation, see there are two positive relationships. So, basically Jenera and Bercy are enemies to each other, but they have a common friend which is Bearland. So, let us see what happens to this. I am closing it and its doing its thing. And yes this is what we had to see. Bearland Jenera and Bercy they all are friends now. So, over time what happens? Jenera and Bercy which were initially enemies they become friends to each other. So, you can you can check more examples as well. So, overall this network is completely stable.

In the next video we are going to visualize these two groups that get formed.