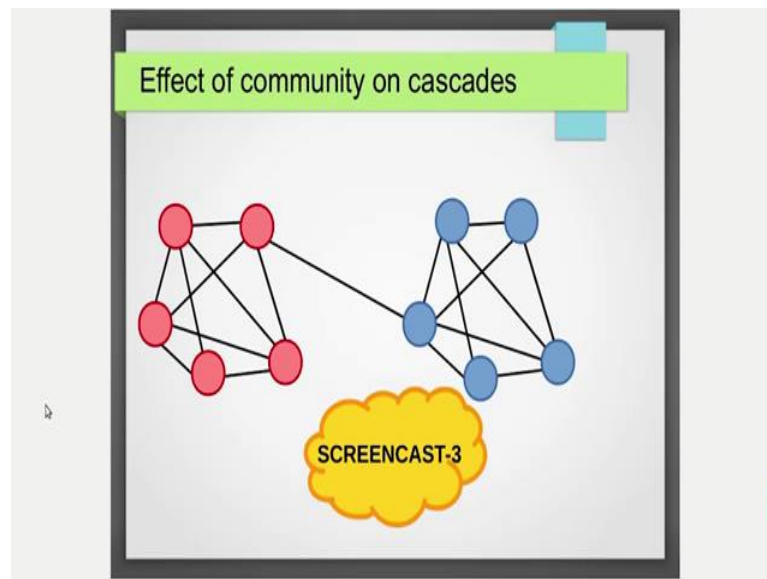


**Social Networks**  
**Prof. S. R. S. Iyengar**  
**Department of Computer Science**  
**Indian Institute of Technology, Ropar**

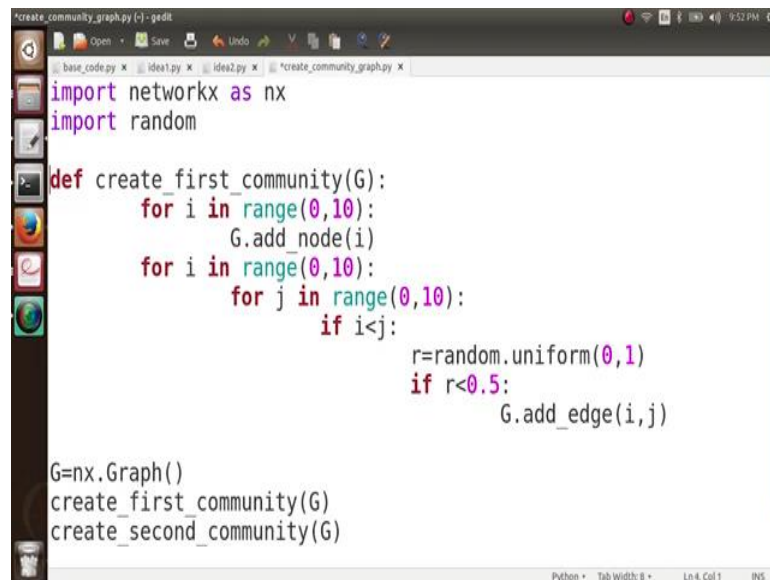
**Cascading Behavior in Networks**  
**Lecture - 99**  
**Coding the Third Big Idea- Impact of Communities on Cascades**

(Refer Slide Time: 00:05)



Now, we are going to talk about the third idea which is very cute which says that if you have many communities in your network. So, for the sake of simplicity, we will take two communities and we show that if our cascade starts from one community; then it actually even if it gets into this entire community, it is difficult for this cascade to get into another community. So, we will be coding and now we will be looking at this aspect.

(Refer Slide Time: 00:31)

A screenshot of a Python IDE window titled 'create\_community\_graph.py (-) - gedit'. The window shows a Python script with the following code:

```
import networkx as nx
import random

def create_first_community(G):
    for i in range(0,10):
        G.add_node(i)
    for i in range(0,10):
        for j in range(0,10):
            if i<j:
                r=random.uniform(0,1)
                if r<0.5:
                    G.add_edge(i,j)

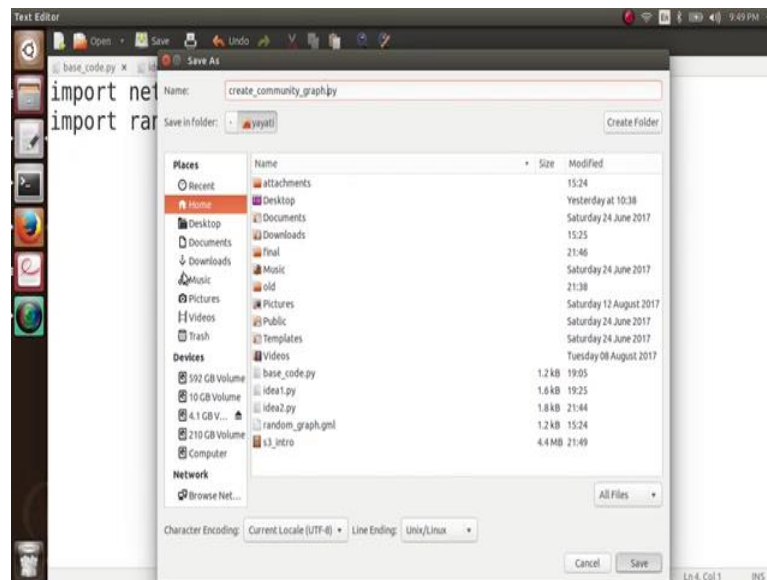
G=nx.Graph()
create_first_community(G)
create_second_community(G)
```

The code defines a function to create a graph with two communities. The first community is created by adding 10 nodes and connecting them in a dense manner. The second community is also created, but its code is not visible in the screenshot. The IDE interface includes a menu bar with 'File', 'Edit', 'View', 'Tools', and 'Help', and a status bar at the bottom showing 'Python', 'Tab Width: 8', 'Ln 4, Col 1', and 'RHS'.

So, for implementing this, we need a graph which consist of two communities: two dense communities. So, what will we do? What will we do? We will create an artificial graph which will be consisting of two communities. How do we create this artificial graph is we will create a graph having 20 nodes? So, first 10 nodes will be connected to each other and second 10 nodes means nodes 0 to 9 will be connected to each other and node 10 to 19 will be connected to each other and there will be just one link between these two communities.

Let us just quickly do it. So, I will just import networkx as nx and I will also need the random functions. So, a import random as well.

(Refer Slide Time: 01:24)



Let me save this as this file as create community graph.py ok. So, I import it random and then what I will do is I create a graph  $G = nx.graph$ . After this I create the first community in this graph. So, I call the function create first community and what does this function do is, it just creates a random links between the people I will just show you. So, define, create, first community and we pass the graph here first community G and what you do here is ok. So, first I add 10 nodes in this community and no edge is. So, for  $i$  in range 0 to 10, what do I do is  $G.add\_node(i)$ . So, I have added 10 nodes in this graph, next I want to add some links.

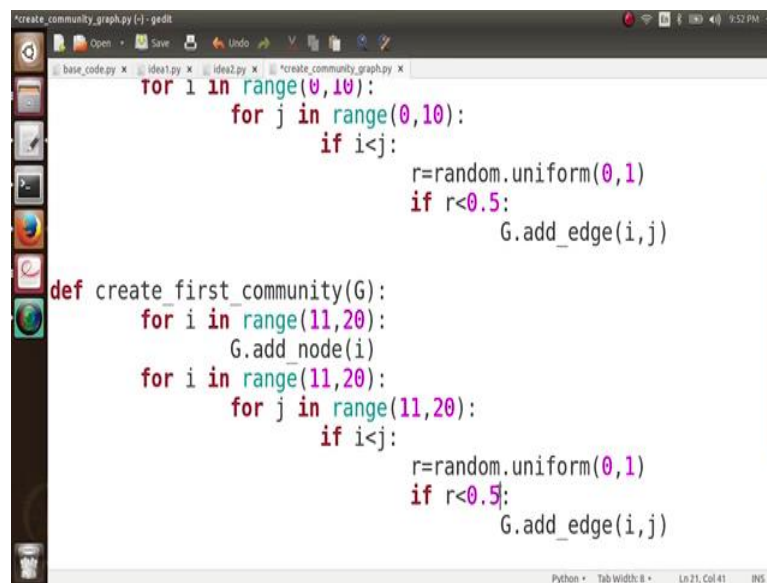
So, how do I add some links? For as we know these are 10 nodes and there are 10 chose to possible edges between these nodes. So, I will take every edge and I will put every edge with the probability of 0.5. So, I am doing exactly what happens in an Erdos-Renyi graph, but I am just coding it manually. So, for  $i$  in range 0 to 0 so, I will put first iterator over all of these nodes and then for  $j$  in range 0 to 10, I put a second iterator here and again to avoid duplication and the same number if  $i < j$  so, that I get every pair only once.

What do I do? I will want to put an edge here with the probability of 0.5 and I am quite sure that you know what I am going to do next because we have seen it previously. We have coded it also previously. If I want to do some event with a probability, I create a random number; random uniform number real number from 0 to 1. So, I will get a real

number from 0 to 1.  $r = \text{random.uniform}(0, 1)$  and then if  $r$  is less than 0.5. What I do is `G.add_edge` from  $i$  to  $j$ .

So, I have added every possible edge between these 10 nodes with the probability of 0.5. So, this is my first community. Similarly, I will create my second community. Create second community and I am sure that you know that the code is going to be same except for the numbering of nodes.

(Refer Slide Time: 04:45)

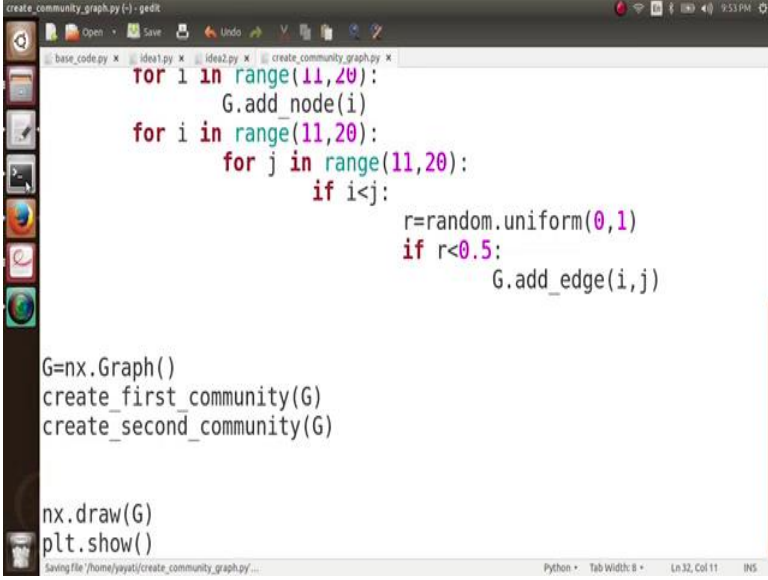
A screenshot of a Python IDE window titled 'create\_community\_graph.py - gedit'. The window shows two code snippets. The first snippet is a nested loop that iterates over nodes 0 to 9 and adds edges between them with a probability of 0.5. The second snippet is a function 'create\_first\_community(G)' that adds nodes 11 to 20 and then adds edges between them with a probability of 0.5. The code is as follows:

```
for i in range(0,10):
    for j in range(0,10):
        if i<j:
            r=random.uniform(0,1)
            if r<0.5:
                G.add_edge(i,j)

def create_first_community(G):
    for i in range(11,20):
        G.add_node(i)
    for i in range(11,20):
        for j in range(11,20):
            if i<j:
                r=random.uniform(0,1)
                if r<0.5:
                    G.add_edge(i,j)
```

So, we can just copy paste this code here and what I can do is I know that my nodes are now going to run from 11 to 20 right. So, from 11 to 20 and here also 11 to 20 and here also 11 to 20, everything is done. So, 10 nodes I created here in the; similar way 10 nodes, I created here and between these 10 nodes. I will again put an edge with the probability of 0.5.

(Refer Slide Time: 05:11)



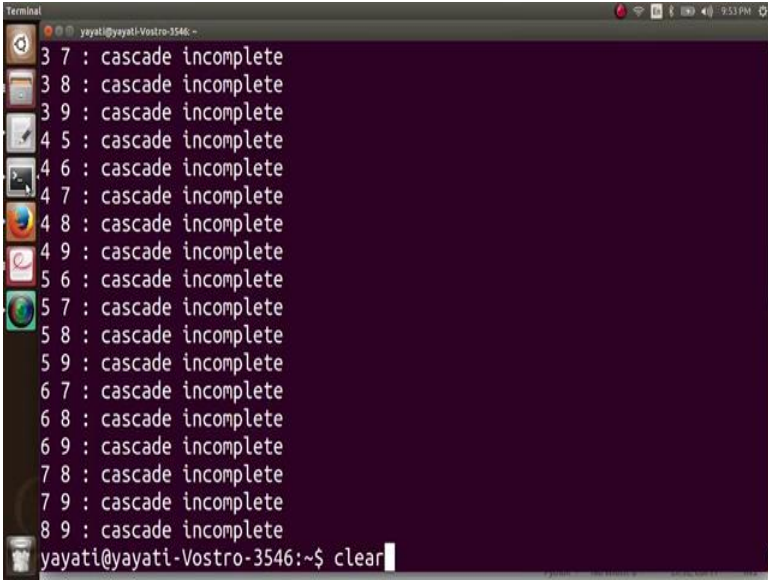
```
create_community_graph.py - gedit
base_code.py x idea1.py x idea2.py x create_community_graph.py x
for i in range(11,20):
    G.add_node(i)
for i in range(11,20):
    for j in range(11,20):
        if i<j:
            r=random.uniform(0,1)
            if r<0.5:
                G.add_edge(i,j)

G=nx.Graph()
create_first_community(G)
create_second_community(G)

nx.draw(G)
plt.show()
Saving file /home/yayati/create_community_graph.py... Python Tab Width: 8 Ln:32, Col:11 INS
```

So, what am I going to have now in my network, there will be two separate communities, or you can say two components in this graph? So, what currently we have made is a disconnected graph. I can also actually show you this graph. So, let me show you this graph `nx.draw (G)`, then we have a `plt.show` right and let us run.

(Refer Slide Time: 05:48)



```
Terminal
yayati@yayati-Vostro-3546: ~
3 7 : cascade incomplete
3 8 : cascade incomplete
3 9 : cascade incomplete
4 5 : cascade incomplete
4 6 : cascade incomplete
4 7 : cascade incomplete
4 8 : cascade incomplete
4 9 : cascade incomplete
5 6 : cascade incomplete
5 7 : cascade incomplete
5 8 : cascade incomplete
5 9 : cascade incomplete
6 7 : cascade incomplete
6 8 : cascade incomplete
6 9 : cascade incomplete
7 8 : cascade incomplete
7 9 : cascade incomplete
8 9 : cascade incomplete
yayati@yayati-Vostro-3546:~$ clear
```

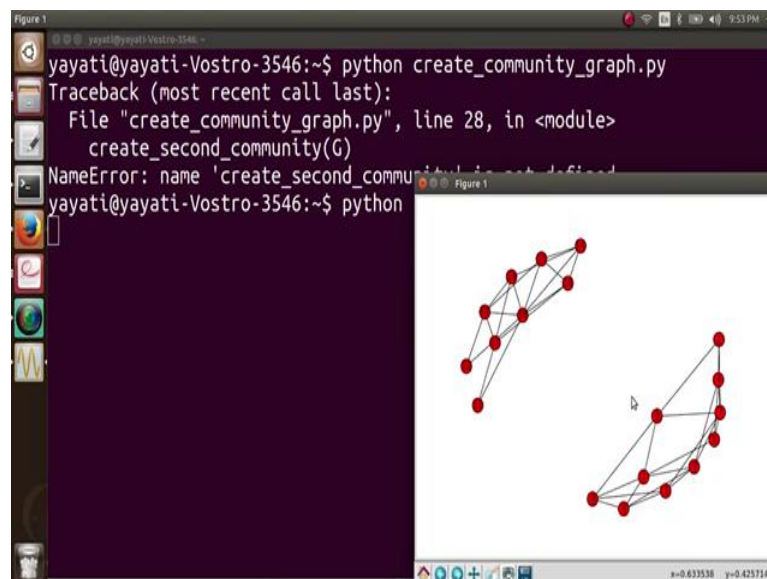
It create community graph.

(Refer Slide Time: 05:52)

```
Terminal
yayati@yayati-Vostro-3546:~$ python create_community_graph.py
Traceback (most recent call last):
  File "create_community_graph.py", line 28, in <module>
    create_second_community(G)
NameError: name 'create_second_community' is not defined
yayati@yayati-Vostro-3546:~$ python create_community_graph.py
yayati@yayati-Vostro-3546:~$ python create_community_graph.py
yayati@yayati-Vostro-3546:~$ python idea3.
```

I am just create.

(Refer Slide Time: 06:01)

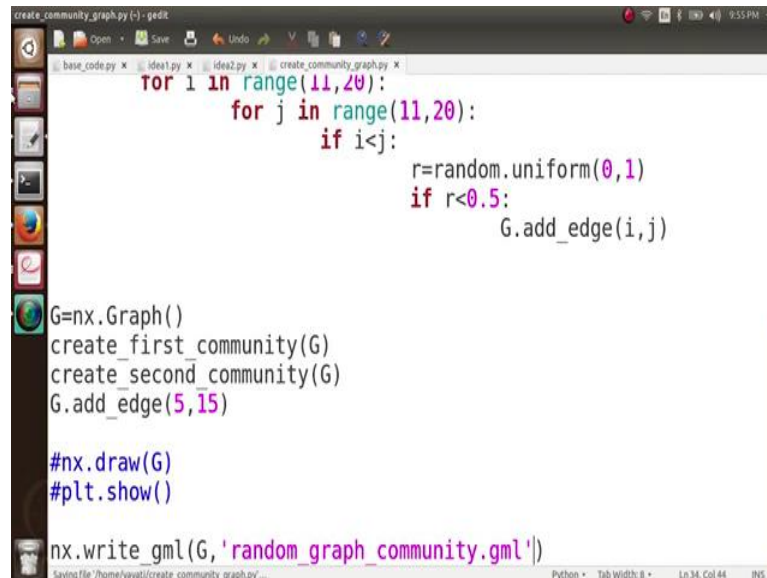


We forgot to change the name of function here. So, it is create second community. So, you can see this graph here right. We have one community here having which is random graph with the 10 nodes and 0.5 probability and one community here.

Now, these are components we cannot say these are communities also, but they are components; we want a connected graph. So, what I am going to put next is I am going

to put just one edge between these two communities. So, that the communities in my graph are very well defined; I put only 1 edge.

(Refer Slide Time: 06:42)



```
create_community_graph.py (-) - gedit
base_code.py x idea1.py x idea2.py x create_community_graph.py x
for i in range(11,20):
    for j in range(11,20):
        if i<j:
            r=random.uniform(0,1)
            if r<0.5:
                G.add_edge(i,j)

G=nx.Graph()
create_first_community(G)
create_second_community(G)
G.add_edge(5,15)

#nx.draw(G)
#plt.show()

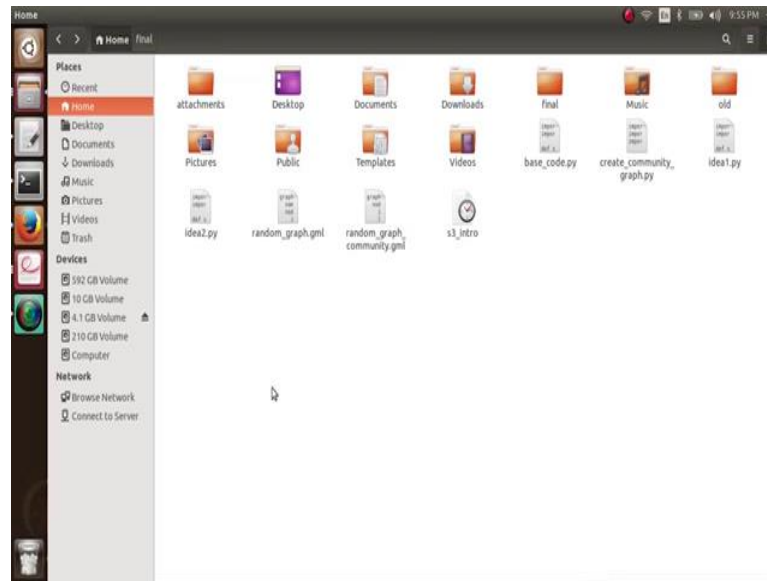
nx.write_gml(G,'random_graph_community.gml')
```

So, I what do I do is G.add\_edge and I add an edge.

Student: (Refer Time: 06:48).

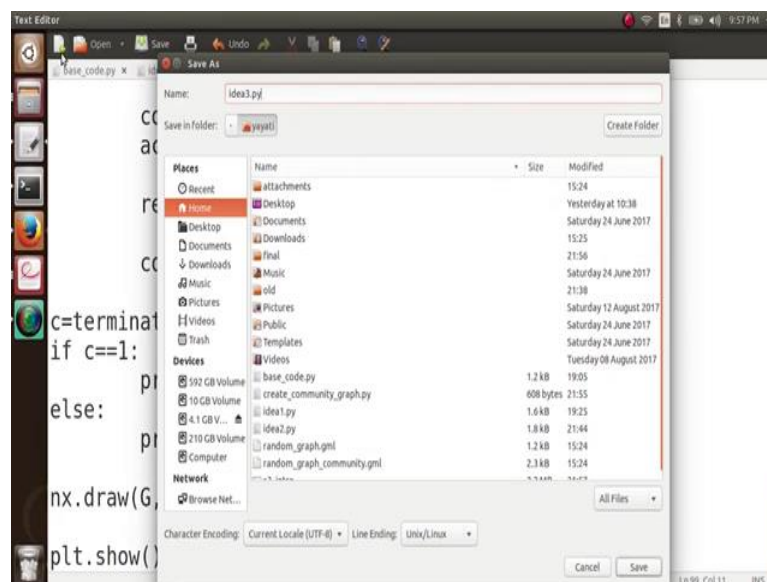
Let us say from node 5 to 15. So, let us run it. So, you can see here now this is my graph having two communities. We are going to work with this graph for a third idea, for the implementation of the third idea. What I will do is, I store this graph nx.write\_gml and I store this graph as let me name it as random graph with community. So, random graph community.sorry.gml write\_gml; I run it.

(Refer Slide Time: 07:41)



So, you can see a graph here random graph community.gml ok. So, now, we are ready to see. So, we have a graph with community, and we are ready to see the impact of cascading on such a network.

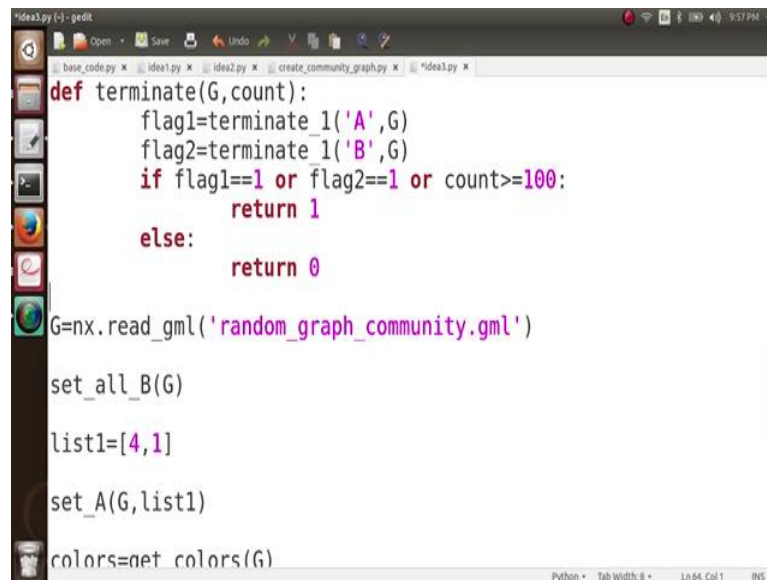
(Refer Slide Time: 08:04)



So, I will actually take my code from here and. Take my code from here and let us save it here. We will do all the changes here. Let us call it idea 3.py.



(Refer Slide Time: 08:13)



```
def terminate(G,count):
    flag1=terminate_1('A',G)
    flag2=terminate_1('B',G)
    if flag1==1 or flag2==1 or count>=100:
        return 1
    else:
        return 0

G=nx.read_gml('random_graph_community.gml')

set_all_B(G)

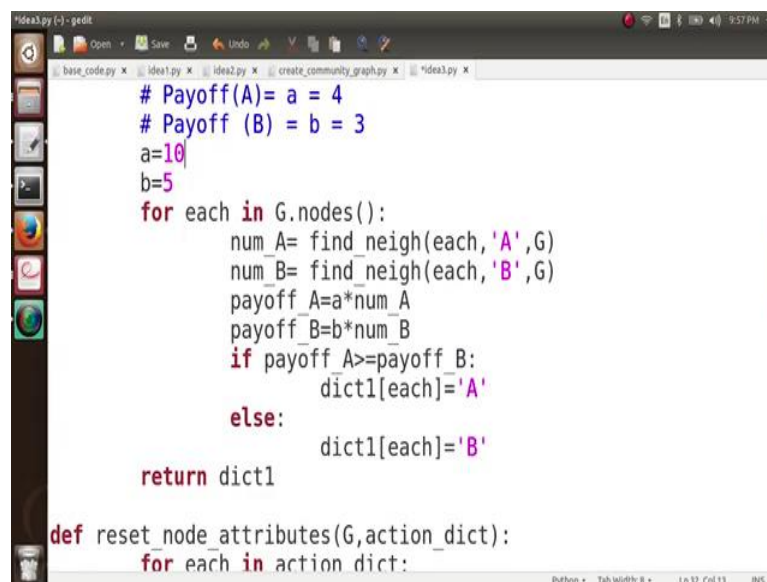
list1=[4,1]

set_A(G,list1)

colors=get_colors(G)
```

So, first of all which is the graph we are going to work with this random underscore graph underscore community one thing and then let us change the payoffs little bit or let us keep it the say.

(Refer Slide Time: 08:26)

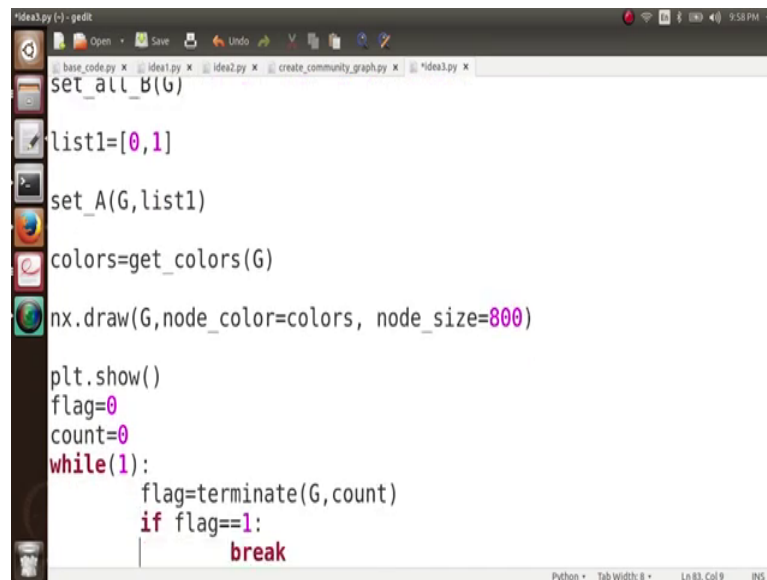


```
# Payoff(A)= a = 4
# Payoff (B) = b = 3
a=10
b=5
for each in G.nodes():
    num_A= find_neigh(each,'A',G)
    num_B= find_neigh(each,'B',G)
    payoff_A=a*num_A
    payoff_B=b*num_B
    if payoff_A>=payoff_B:
        dict1[each]='A'
    else:
        dict1[each]='B'
return dict1

def reset_node_attributes(G,action_dict):
    for each in action_dict:
```

So, let the payoffs be 10 and 5 and let my starting nodes be 0 and 1.

(Refer Slide Time: 08:36)

A screenshot of a Python IDE window titled 'idea3.py (-) - gedit'. The window shows a Python script with the following code:

```
set_all_b(G)

list1=[0,1]

set_A(G,list1)

colors=get_colors(G)

nx.draw(G,node_color=colors, node_size=800)

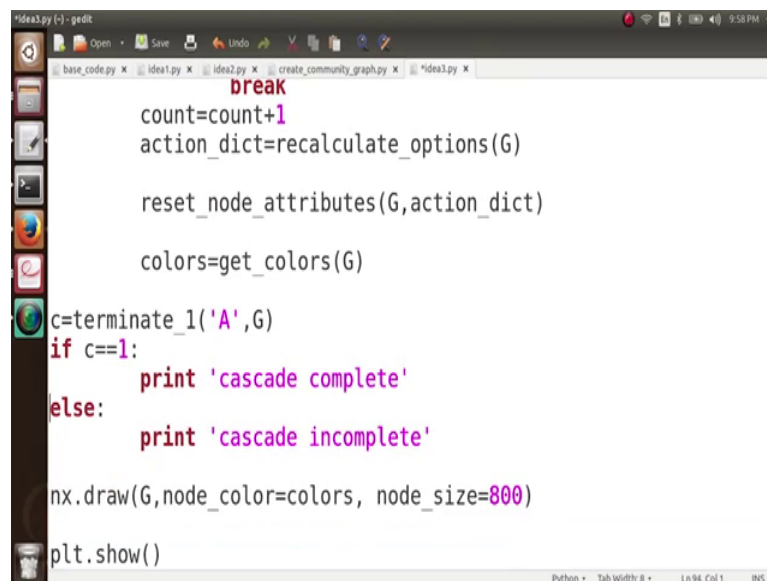
plt.show()
flag=0
count=0
while(1):
    flag=terminate(G,count)
    if flag==1:
        break
```

The status bar at the bottom indicates 'Python • Tab Width: 8 • Ln 83, Col 9 INS'.

```
idea3.py (-) - gedit
base_code.py x idea1.py x idea2.py x create_community_graph.py x idea3.py x
set_all_b(G)
list1=[0,1]
set_A(G,list1)
colors=get_colors(G)
nx.draw(G,node_color=colors, node_size=800)
plt.show()
flag=0
count=0
while(1):
    flag=terminate(G,count)
    if flag==1:
        break
Python • Tab Width: 8 • Ln 83, Col 9 INS
```

And what I am interested in is looking at how will this cascade going to occur.

(Refer Slide Time: 08:42)

A screenshot of a Python IDE window titled 'idea3.py (-) - gedit'. The window shows a Python script with the following code:

```
        break
count=count+1
action_dict=recalculate_options(G)

reset_node_attributes(G,action_dict)

colors=get_colors(G)

c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'

nx.draw(G,node_color=colors, node_size=800)

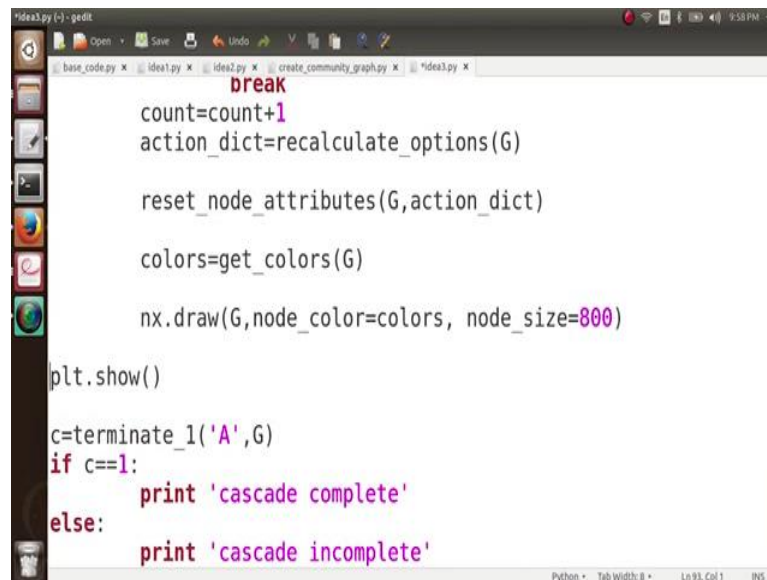
plt.show()
```

The status bar at the bottom indicates 'Python • Tab Width: 8 • Ln 94, Col 1 INS'.

```
idea3.py (-) - gedit
base_code.py x idea1.py x idea2.py x create_community_graph.py x idea3.py x
        break
count=count+1
action_dict=recalculate_options(G)
reset_node_attributes(G,action_dict)
colors=get_colors(G)
c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'
nx.draw(G,node_color=colors, node_size=800)
plt.show()
Python • Tab Width: 8 • Ln 94, Col 1 INS
```

And let me look at this in every step.

(Refer Slide Time: 08:48)



```
break
count=count+1
action_dict=recalculate_options(G)

reset_node_attributes(G,action_dict)

colors=get_colors(G)

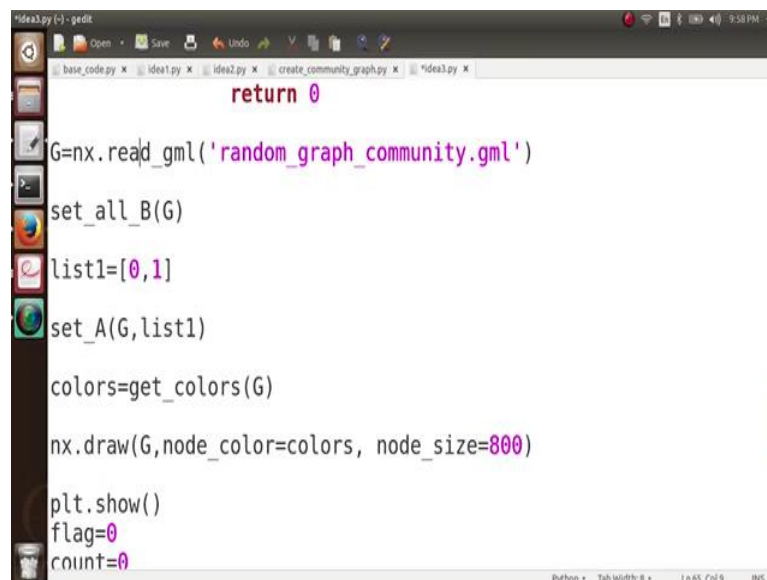
nx.draw(G,node_color=colors, node_size=800)

plt.show()

c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'
```

I want to see the graph at every step. So, you can see I will just show you the code. It is the same code which we have written previously.

(Refer Slide Time: 08:58)



```
return 0

G=nx.read_gml('random_graph_community.gml')

set_all_B(G)

list1=[0,1]

set_A(G,list1)

colors=get_colors(G)

nx.draw(G,node_color=colors, node_size=800)

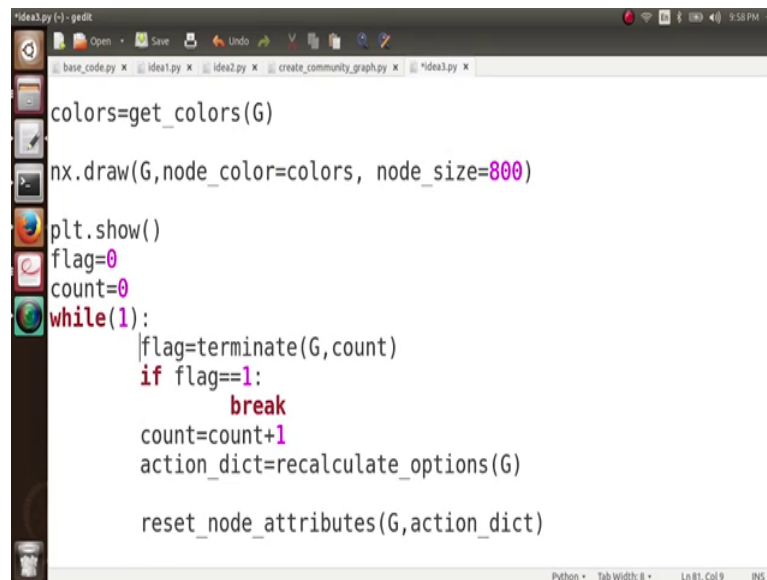
plt.show()

flag=0

count=0
```

So, here we load a graph. Here we set all the nodes to have the action B, here we choose the initial adopters. We set the initial adopters, we draw the graph.

(Refer Slide Time: 09:09)

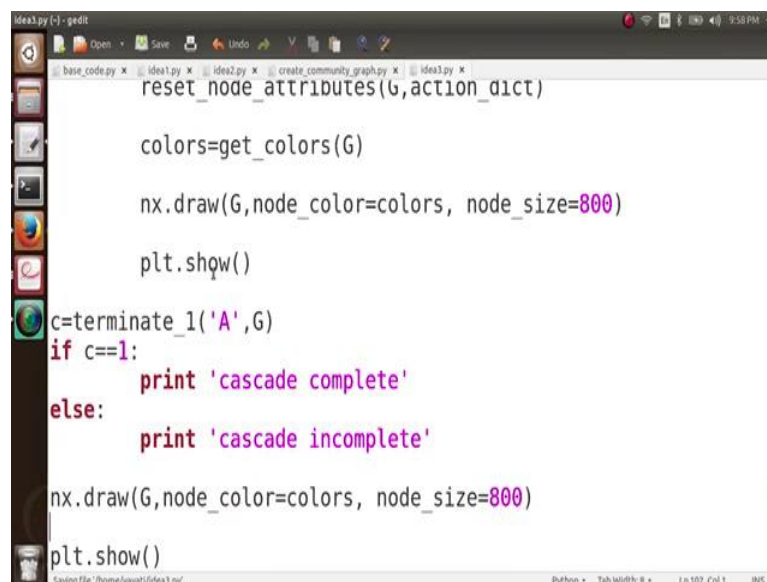
A screenshot of a Python IDE window titled 'idea3.py (-) - gedit'. The window contains several tabs: 'base\_code.py', 'idea1.py', 'idea2.py', 'create\_community\_graph.py', and 'idea3.py'. The active tab 'idea3.py' displays the following Python code:

```
colors=get_colors(G)
nx.draw(G,node_color=colors, node_size=800)
plt.show()
flag=0
count=0
while(1):
    flag=terminate(G,count)
    if flag==1:
        break
    count=count+1
    action_dict=recalculate_options(G)
    reset_node_attributes(G,action_dict)
```

The status bar at the bottom indicates 'Python • Tab Width: 8 • Ln 81, Col 9 • INS'.

Next, this is just a while loop in which are process runs again and again. Terminating condition, as we have discussed before action dictionary, we calculate the next snapshot and then we upload the next snapshot exactly what we want to do.

(Refer Slide Time: 09:25)

A screenshot of a Python IDE window titled 'idea3.py (-) - gedit'. The window contains several tabs: 'base\_code.py', 'idea1.py', 'idea2.py', 'create\_community\_graph.py', and 'idea3.py'. The active tab 'idea3.py' displays the following Python code:

```
reset_node_attributes(G,action_dict)

colors=get_colors(G)

nx.draw(G,node_color=colors, node_size=800)

plt.shqw()

c=terminate_1('A',G)
if c==1:
    print 'cascade complete'
else:
    print 'cascade incomplete'

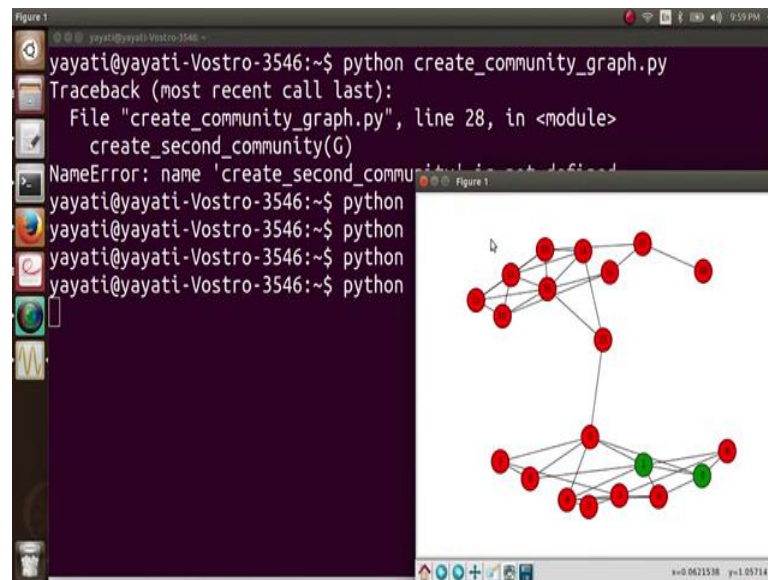
nx.draw(G,node_color=colors, node_size=800)

plt.show()
```

The status bar at the bottom indicates 'Python • Tab Width: 8 • Ln 102, Col 1 • INS'.

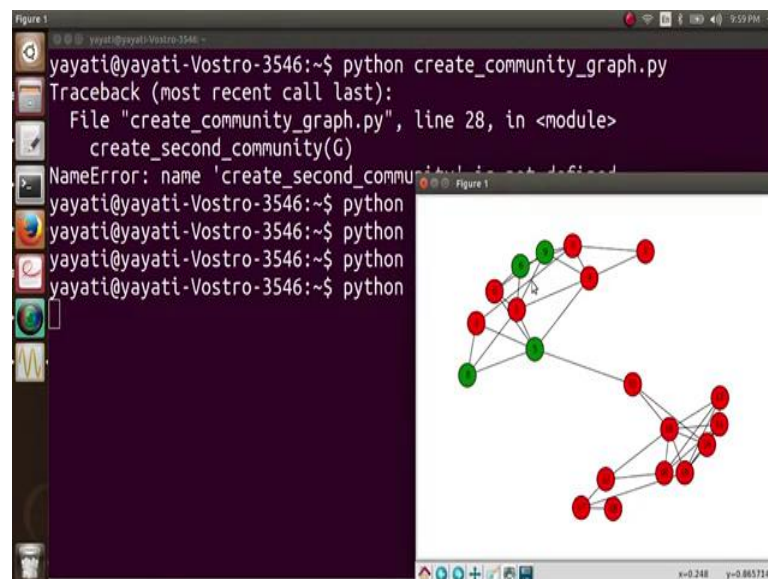
So, we just the code is very simple, we just have to run this code and see what happens. So, let us implement it let us execute it python idea 3.py ok.

(Refer Slide Time: 09:37)



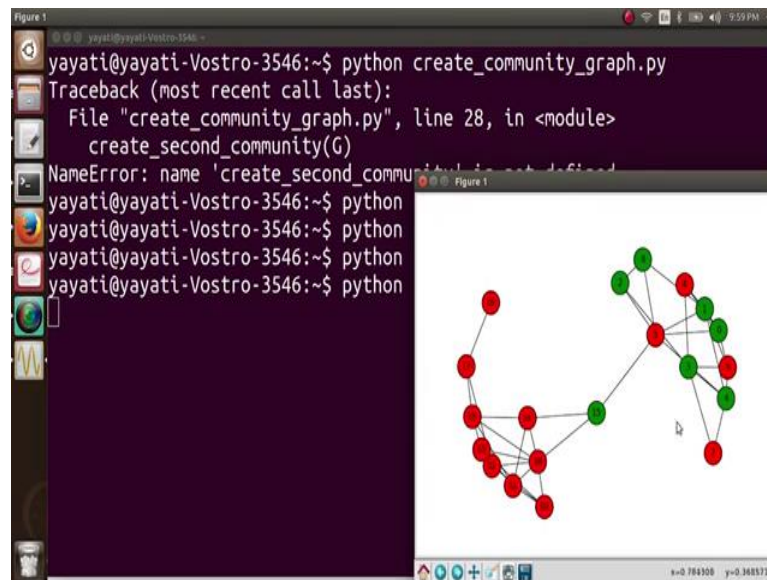
So, this is your initial graph and the nodes 0 and 1 are infected here.

(Refer Slide Time: 09:44)



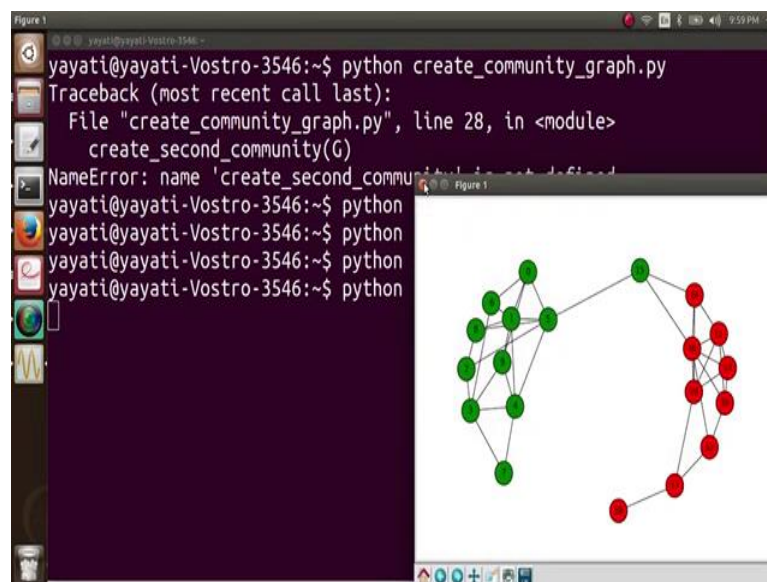
I have adopted the idea and then you can see that 0 and 1 have adopted back the behavior B, but these other four nodes in this community have adopted this behavior A.

(Refer Slide Time: 09:56)



And then you see more nodes have adopted this behavior A and this one node in this community have also adopted this behavior A and then you can see that most of the nodes in this first community where the cascade started have adopted the behavior A.

(Refer Slide Time: 10:12)



And then you see, one community has completely adopted the behavior A while in other communities still everybody has adopted just the behavior B. And you can keep running it again and again and you can see that this behavior you not passing on to the other

community. It remains stuck to this first community only. We can see that this cascade it keeps strapped in the first community itself.

So, this graph is not going to change, it will just keep running for hundred iterations like this and your cascade has actually come to an end. So, if you remember during the lecture we discussed that there is this, there are three stability conditions: first is everybody in this network adopts A, second is everybody in this network adopts B and third is some nodes in this network adopt A while others adopt B. So, you can see that is the third condition here.

Some nodes in this network have adopted A, other nodes in this network have adopted B and then we have looked at what should the network look like for the third condition to happen and that was the presence of communities and that is exactly what is happening here. Here are two communities one community has adopted one behavior and the second community has adopted a different behavior.