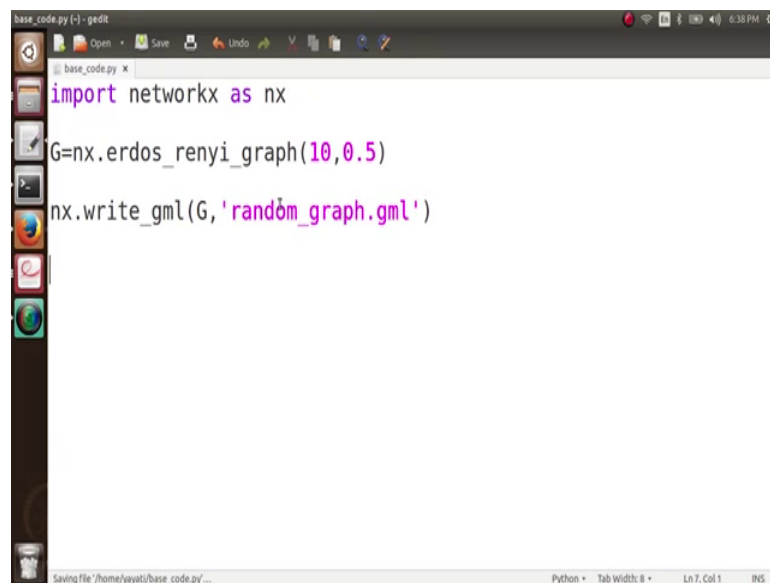


Social Networks
Prof. S. R. S. Iyengar
Department of Computer Science
Indian Institute of Technology, Ropar

Cascading Behavior in Networks
Lecture - 96
The Base Code

(Refer Slide Time: 00:05)



```
base_code.py (-) - gedit
base_code.py x
import networkx as nx
G=nx.erdos_renyi_graph(10,0.5)
nx.write_gml(G,'random_graph.gml')
```

The screenshot shows a gedit window titled 'base_code.py (-) - gedit'. The code in the editor is: `import networkx as nx`, `G=nx.erdos_renyi_graph(10,0.5)`, and `nx.write_gml(G,'random_graph.gml')`. The status bar at the bottom indicates 'Python', 'Tab Width: 8', 'Ln 7, Col 1', and 'INS'.

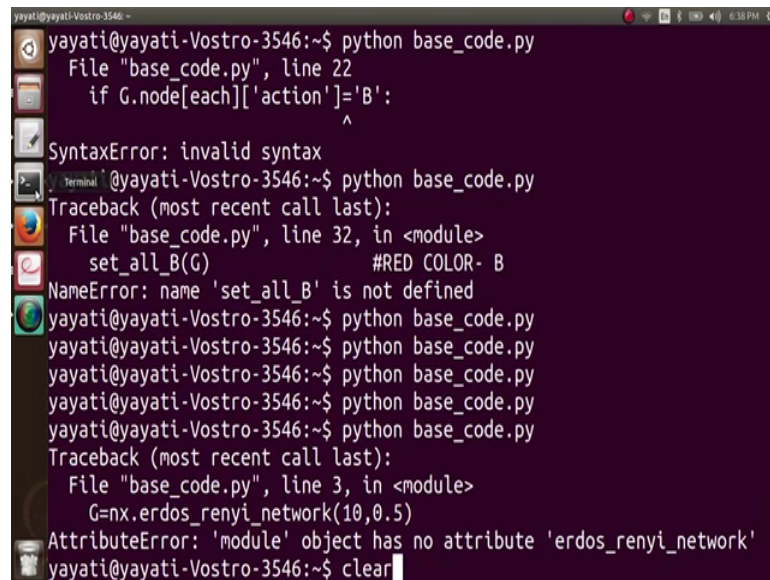
So, we have a file here. So, I am using this file base code dot py for the very first code. What we are first going to do is as and always import networkx as nx since we want to work on graphs. Next what we need is an underline network. So, for implementing anything how is action a diffusing on this network or action b diffusing on this network, how is the cascade going.

So, for everything I need an underline graph. And I am going to take this underline graph as an Erdos Renyi graph as a random graph. So, I make here Erdos Renyi graph G equals to nx dot erdos renyi graph and then I want this graph to have 10 nodes.

So, first parameter is the number of nodes which is 10 and the second parameter is the probability with which an edge is represent, which is 0.5 And what I will also do is I want to use the same graph for some reasons I want to use the same underline graph for the coming up codes.

So, what I am going to do is, i will do G, I am going to store this graph as a gml file, so I use nx dot write gml and I write this graph G as random graph dot gml random underscore graph dot gml And we will be using this graph further let us just execute it.

(Refer Slide Time: 01:43)

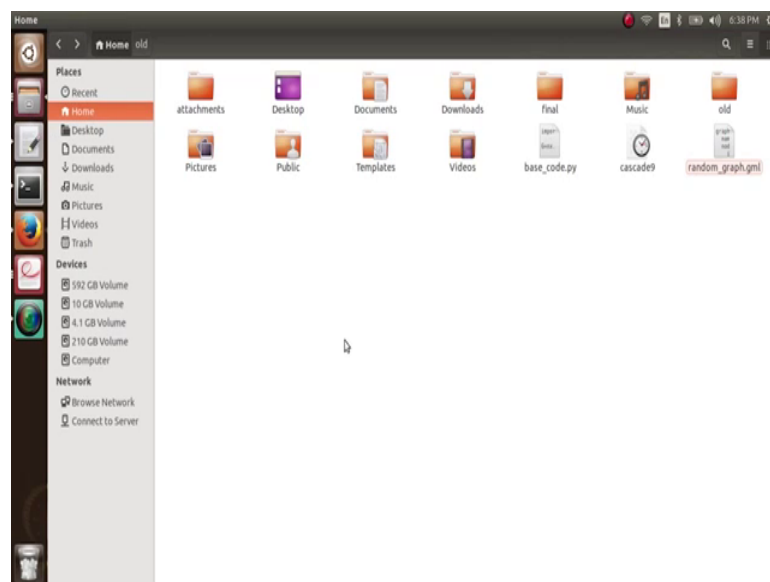


```
yayati@yayati-Vostro-3546:~$ python base_code.py
File "base_code.py", line 22
    if G.node[each]['action']='B':
        ^
SyntaxError: invalid syntax

Terminal yayati-Vostro-3546:~$ python base_code.py
Traceback (most recent call last):
  File "base_code.py", line 32, in <module>
    set_all_B(G) #RED COLOR- B
NameError: name 'set_all_B' is not defined

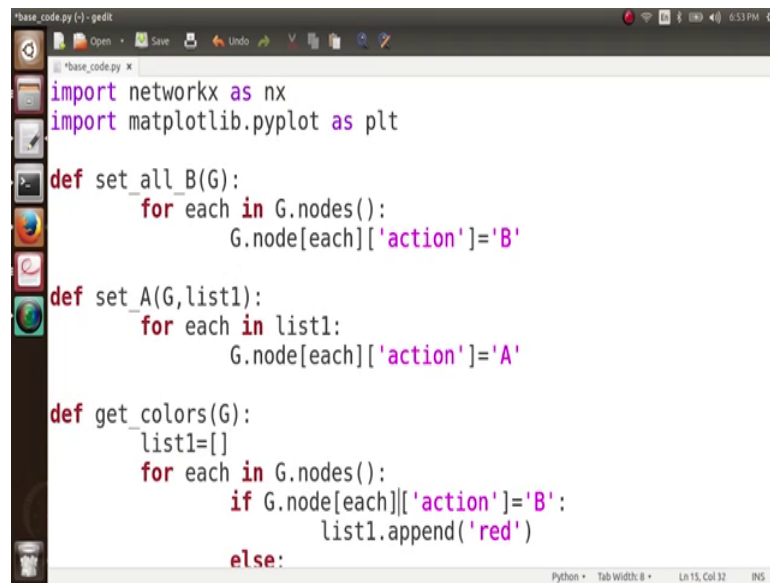
yayati@yayati-Vostro-3546:~$ python base_code.py
yayati@yayati-Vostro-3546:~$ python base_code.py
yayati@yayati-Vostro-3546:~$ python base_code.py
yayati@yayati-Vostro-3546:~$ python base_code.py
yayati@yayati-Vostro-3546:~$ python base_code.py
Traceback (most recent call last):
  File "base_code.py", line 3, in <module>
    G=nx.erdos_renyi_network(10,0.5)
AttributeError: 'module' object has no attribute 'erdos_renyi_network'
yayati@yayati-Vostro-3546:~$ clear
```

(Refer Slide Time: 01:50)



So we have executed it and you can see that here we have these graph random underscore graph dot gml. So, now, on we are using only this graph random underscore graph dot gml ok.

(Refer Slide Time: 02:01)

A screenshot of a code editor window titled 'base_code.py (-) - gedit'. The code is as follows:

```
import networkx as nx
import matplotlib.pyplot as plt

def set_all_B(G):
    for each in G.nodes():
        G.node[each]['action']='B'

def set_A(G,list1):
    for each in list1:
        G.node[each]['action']='A'

def get_colors(G):
    list1=[]
    for each in G.nodes():
        if G.node[each]['action']='B':
            list1.append('red')
        else:
```

The editor interface includes a menu bar with 'Open', 'Save', 'Undo', and 'V' options. A sidebar on the left shows various system icons. The status bar at the bottom indicates 'Python', 'Tab Width: 8', 'Ln 15, Col 32', and 'ING'.

So, I have remove the other statements from this code we have already run it once. So, we have that graph, random underscore graph stored in our system. So, first of all I will load that graph. So, what I do is G equals to n x dot read gml and then I have the graph here, random underscore graph dot gml, I have this graph here. Next what we want to do is, we want to implement the concept that initially every node in this network is having an action B associated with it and then the action A is introduced. How do you introduce action A is, by some nodes in this network flipping their status from B to A.

So, implementing that is very easy. We can implement it with the help of attributes associated with the nodes. So, initially what we will do is every node will have an attribute action and the value of this attribute action for every node will be equal to B So, we call a function for that is set all B G; which mean that in your graph G for all the nodes set its action to be equals to B and then we can quickly define this function here.

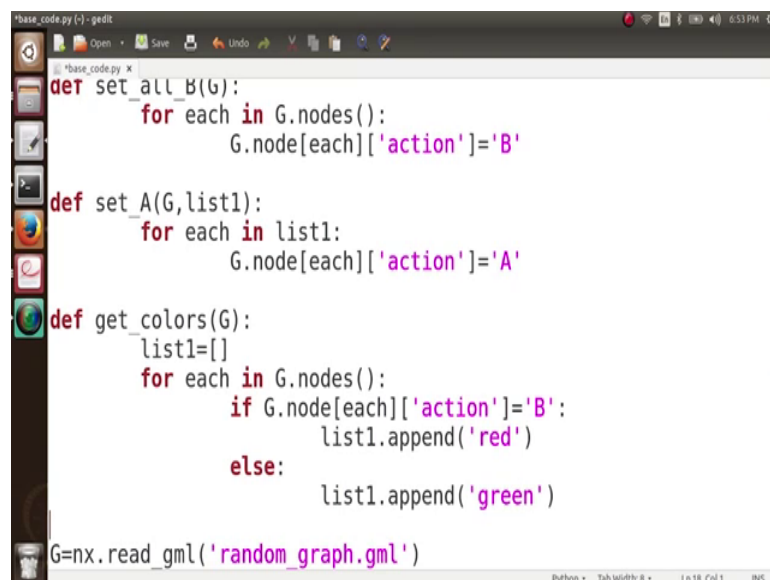
Define set all BG and how do we define it now we get an iterator for each in G dot nodes for each in G dot nodes what we have to do is set the attribute action associated with this node to be equal to B. So, what do we do is G dot node each and then what is the action associated with this node it is equal to B.

So, all the nodes in this network currently has this action B. Next what I want to do is I will pick some 2 nodes from this network and for those 2 nodes I want to set the action associated with them to be A. So, how do I do that? One way to do that is I will take a

list here and this list here consists of some 2 nodes. So, let us say 3 and 7 So, I want these 2 nodes 3 and 7 in this network to be the nodes which initially adopt action A. So I want to change the attributes corresponding to these nodes, so what do I call is set A G comma list 1.

So, whatever is there in list 1, what all nodes are there in list 1 I want to set their attribute to be equals to A. And that is again quiet simple define set A and then I have a G here and I have a list 1 here and what do I do is for each in list 1 G dot node each action and what is this action is going to be is A.

(Refer Slide Time: 05:23)



```
def set_all_B(G):
    for each in G.nodes():
        G.node[each]['action']='B'

def set_A(G, list1):
    for each in list1:
        G.node[each]['action']='A'

def get_colors(G):
    list1=[]
    for each in G.nodes():
        if G.node[each]['action']='B':
            list1.append('red')
        else:
            list1.append('green')

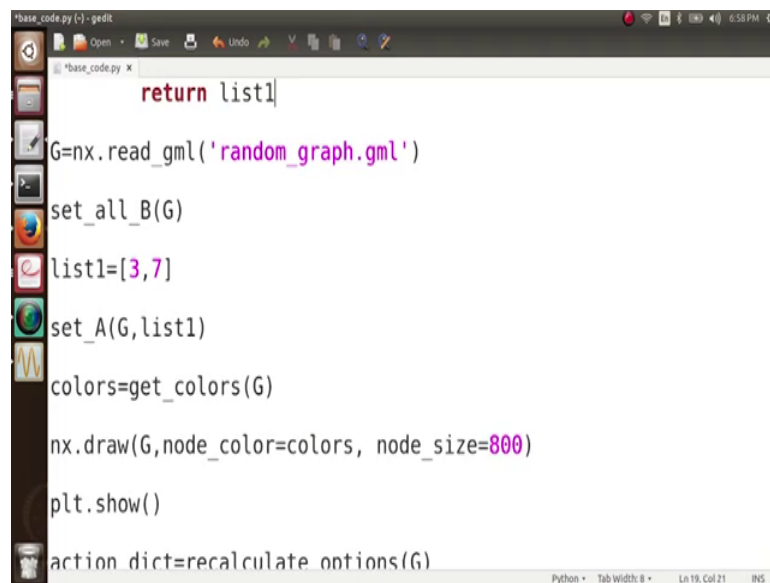
G=nx.read_gml('random_graph.gml')
```

So, I have a graph here in which all the nodes have adopted the action B except these 2 nodes which are the initial adopters 3 and 7 which are the initial adopters and I have adopted this action A and next I want to visualise this graph. So, visualising this graph is easy and let us visualise it with colours.

So, we have done it previously as well. So, let us associate it a colour red with the action B and the colour green with the action A. So, initially all the nodes in this network will be red and this green behavior will be coming and trying to spread on this network. So, for that I will need an array colours. So, let me get this array from the function get colours G and then this is simple.

So, I have a function here define colours G and what this function is going to do is I have a list here and for each in G dot nodes what we are going to do is if G dot node each and if the action associated with this particular node is B, which means that it should have a red colour. So, list one dot append, red else list one node append green.

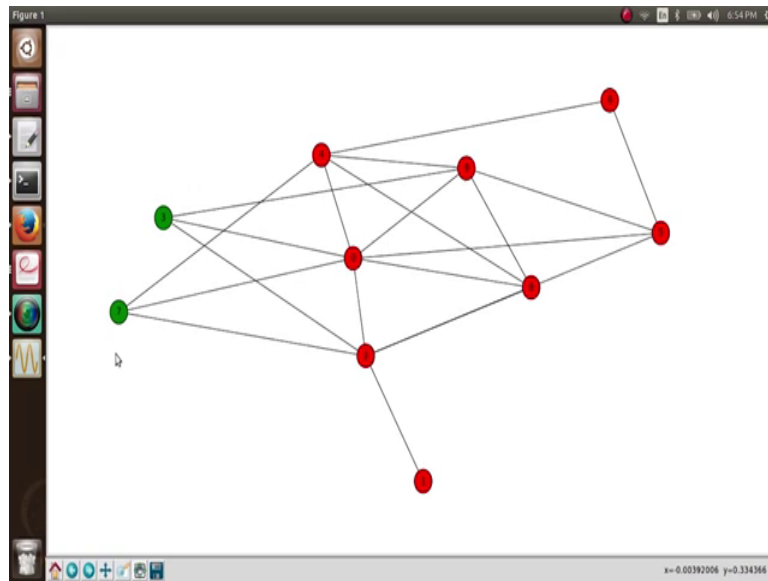
(Refer Slide Time: 07:15)



```
base_code.py (-) - gedit
base_code.py x
return list1
G=nx.read_gml('random_graph.gml')
set_all_B(G)
list1=[3,7]
set_A(G,list1)
colors=get_colors(G)
nx.draw(G,node_color=colors, node_size=800)
plt.show()
action_dict=recalculate_options(G)
```

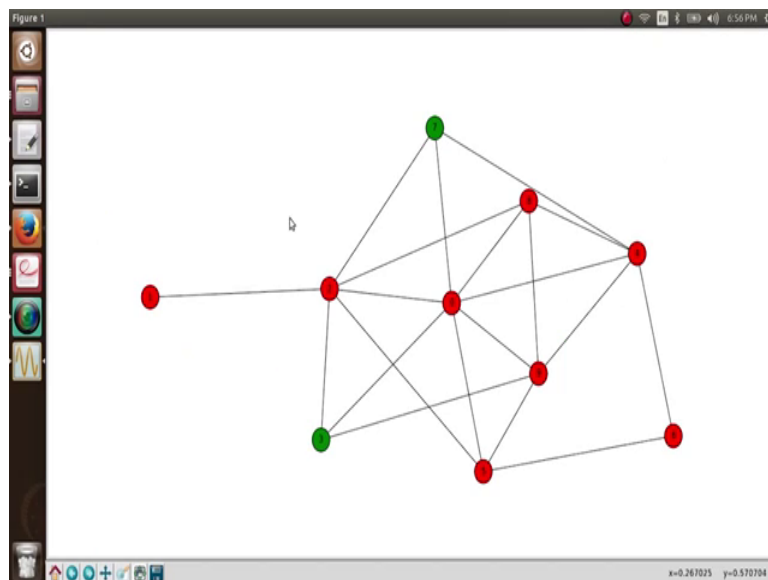
And then return list 1, so, we have an array of colours here. Let us quickly visualise this graph. So, for visualise this visualising this graph we have. So, first of all we need to import the matplotlib; import matplotlib dot pyplot as plt then come down then what we do is nx dot draw G and what should be the node colour; node colour should be taken from the array colours and what should be the node size? Let us see the size of every node to be 800, so that it is be and then we do plt dot show. Let us now turn this piece of code and see. A small mistake in line 15 which should be equals to equals to go back run it ok.

(Refer Slide Time: 08:18)



So, you can see here there is a network and in this network initially all the nodes they were green they had adopted this behavior B and then there are these two nodes three and seven which have been initially decided to adopt the behavior A.

(Refer Slide Time: 08:39)



So, we have the graph here. Next what we want to do is if we look at this graph here you see what is going to, you see what is going to happen next this nodes 7 and 3 each node here from 0 to 9 we will look at their neighbours, we will look at their neighbours and

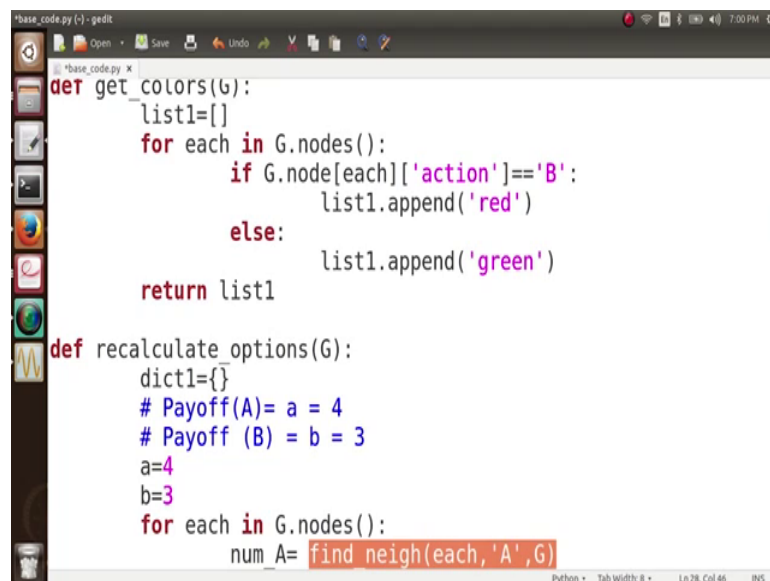
look at the action their neighbours are taking the payoffs and according to that re way their option; for example, this node 2 here looks at all of its neighbours.

So, first of all it looks at its, look at its neighbours who have adopted action B. So, it looks at 1, 8, 0 and 5, so 4 nodes and then let us say the payoff associated with the action B is let us say 2, so 4 into 2 8. 8 is the payoff it gets from here and then it looks at these 2 nodes 7 and 3 and then let us say the payoff associated with this action A is 3. So, gets a payoff of 6 from here. So, it decides to remain in this state B only.

And similarly every node is going to re way this option and according to that the cascade is going to occur is this network. So, let us try to code that. So, what is now going to happen is every node is going to re way its options based on its neighbours, based on their actions and based on the payoffs of those actions as we have seen before. So, we are going to define a function now recalculate options. So, this function recalculate options will one by one look at every node in the network, look at re way its options and it will put the next it will put the decision of this node in a dictionary.

So, we get a dictionary here action dictionary. So, what is this action dictionary? It is a dictionary where keys are the nodes and the values are the actions associated with these nodes. And these dictionary we format from the function recalculate options in G.

(Refer Slide Time: 10:55)



```
def get_colors(G):
    list1=[]
    for each in G.nodes():
        if G.node[each]['action']=='B':
            list1.append('red')
        else:
            list1.append('green')
    return list1

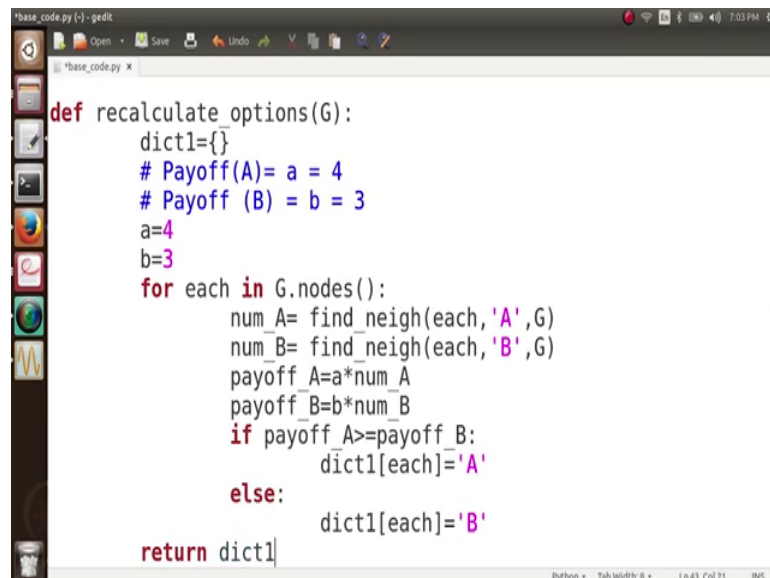
def recalculate_options(G):
    dict1={}
    # Payoff(A)= a = 4
    # Payoff (B) = b = 3
    a=4
    b=3
    for each in G.nodes():
        num_A= find_neigh(each,'A',G)
```

Let us see how does it work. So, we define the function here recalculate options in G and we have a dictionary here let us name it dict1, this is the dictionary which will hold the decision for every node. Now, before moving in further we need to decide the payoffs associated with the action.

So, let us say Payoff associated with the action A. So, let us call it small a and let us take its value to be 4 and then we have a Payoff associated with B let us say b and let us say value to be equal to 3. So, based on these payoffs so, a equals to 4 and b equals to 3, these are the payoffs associated with the 2 actions. Now for each in G dot nodes, but this node is going to do is. First of all this node needs to know how many of its neighbours have adopted the action A, how many of neighbours its neighbours have adopted the action B.

So, let us say number of neighbours who have adopted the action A equals to and we call a function here find neighbours and we pass 2 parameters to this function c and G. So, c is basically c we pass it A and G. So, A is the action which we want to see how many of its neighbours have adopted this action A and G and we want to pass this node also, so each A and G. So, let us now define this function.

(Refer Slide Time: 12:58)



```
def recalculate_options(G):
    dict1={}
    # Payoff(A)= a = 4
    # Payoff (B) = b = 3
    a=4
    b=3
    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
```

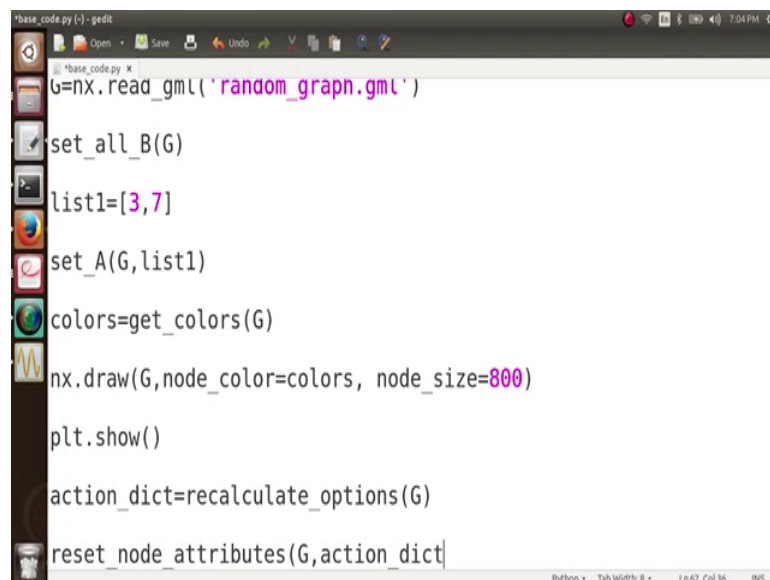
Define and instead of A, we have a character here. So, what this function is going to return in the graph G, the node each we are going to look at the neighbours of the node each and we are going to see how many of these neighbours have adopted this action c.

So, what do we do it is simple we take num to be equal to 0 and then for each I am sorry for each one in G dot neighbours G dot neighbours each if G dot node each one and the action associated with this node is c. Then what we will do is num equals to num plus 1 and then we return this number simple right. So, we have this num A and similarly we can find how many of its neighbours have adopted B. So, find n e i g h each and then we have this B and then we have this G right.

Next what is the total payoff this node gets from adopting the behaviour A. It is nothing, but the payoff associated with this behaviour which is A multiplied by the number of its neighbours which have adopted this behaviour A and similarly what is the payoff it gets from adopting the behaviour B is nothing, but b multiplied by the number of neighbours which have adopted B.

Now this node decides can easily decide whether it wants to adopt the behaviour A or it wants to adopt the behaviour B. So, we know that if payoff associated with A is greater than or equal to payoff with B, what is going to happen is dict 1 for this node which is each is going to be A, it is going to adopt A else dict 1 with each is going to be B and then we return dict 1.

(Refer Slide Time: 15:29)

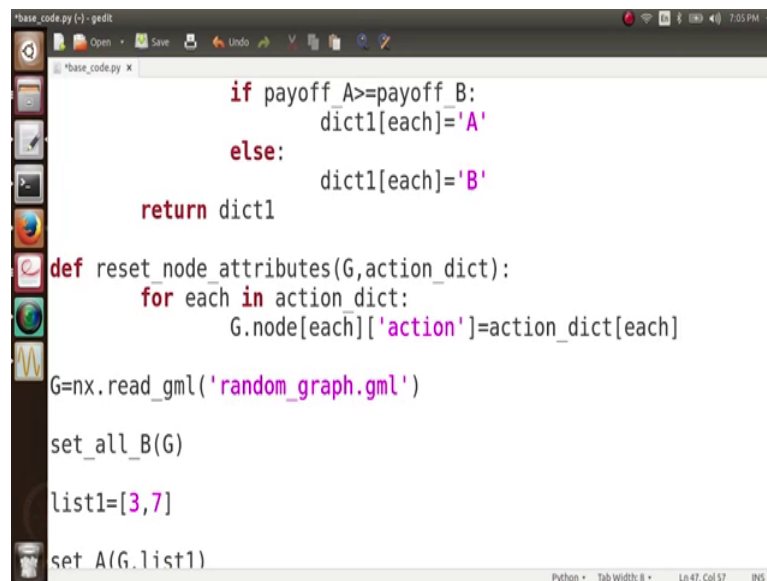


```
base_code.py (-) - gedit
base_code.py
G=nx.read_gml('random_graph.gml')
set_all_B(G)
list1=[3,7]
set_A(G,list1)
colors=get_colors(G)
nx.draw(G,node_color=colors, node_size=800)
plt.show()
action_dict=recalculate_options(G)
reset_node_attributes(G,action_dict
```

So, we get here this action dictionary which tells me what is the decision for the next snapshot that the node makes. So, you see here this action dictionary it is nothing, but it tells me a next snapshot of the graph. It tells me in the next a snapshot in the next

iteration in this process what action every node would have adopted. And now we simply need to draw this graph. So, we have this snapshot stored in action dictionary. So, we want the for the next snapshot we change the attribute associated with the associated with each node based on this action dictionary. So, we do reset node attributes in G and which is based on your action dictionary right action dictionary.

(Refer Slide Time: 16:25)

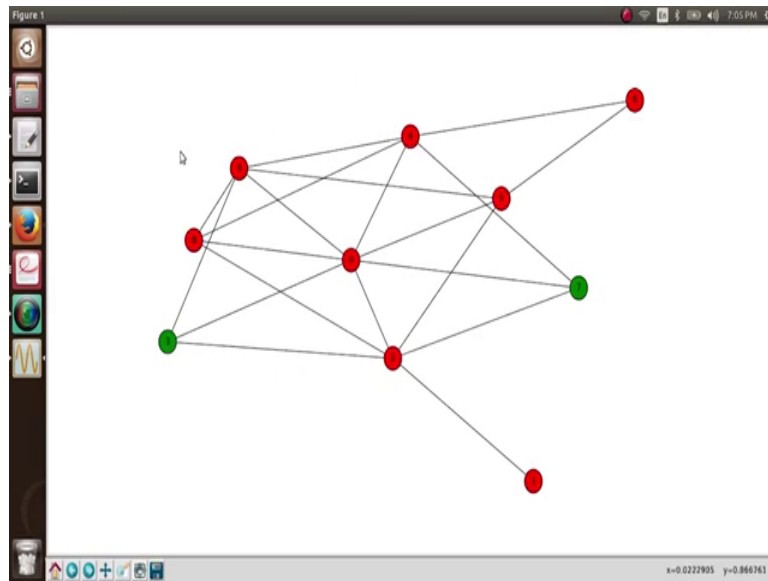


```
base_code.py (-) - gedit
base_code.py x
    if payoff_A >= payoff_B:
        dict1[each] = 'A'
    else:
        dict1[each] = 'B'
    return dict1
def reset_node_attributes(G, action_dict):
    for each in action_dict:
        G.node[each]['action'] = action_dict[each]
G = nx.read_gml('random_graph.gml')
set_all_B(G)
list1 = [3, 7]
set_A(G, list1)
```

We define it here define reset node attributes and what are the parameters G and action dictionary let us say G and action dictionary would we are going to do here if it is simple. So, for each in action dictionary for each for every key in action dictionary G dot node each action, what is it going to be the value associated with that which is action dictionary each is the action.

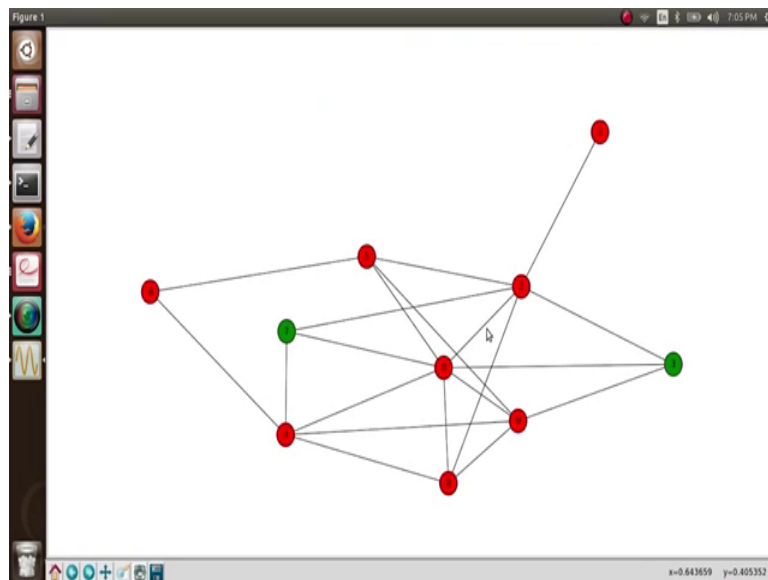
So, we have reset the node attributes here. So, our graph is ready and we just have to visualise this graph and what do we need for visualise, we need to set the right colours of nodes and we need to draw this graph. Just simply copy and paste this node here; let us now execute it and see.

(Refer Slide Time: 17:31)



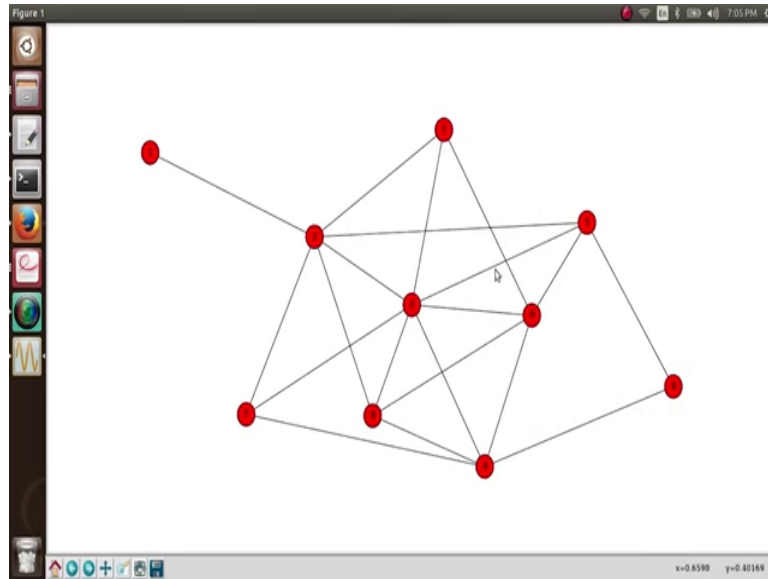
So, you can see here this is the initial graph where our nodes 7 and 3 they have adopted the behaviour A and just small mistake in the spelling of neighbours ok.

(Refer Slide Time: 18:02)



So, this is the initial graph where these nodes 7 and 3 they were they have adopted the behaviour A and you see what happened in next iteration.

(Refer Slide Time: 18:08)



Everybody reviewed their option and these node 2 nodes initially 3 and 7 which had adopted which had decided to adopt this behaviour A, they also backed off and they went back to this behaviour B. We have seen that although the payoff associated with A was high it was 4 and with B was 3, but everybody has switched back to the same condition of having adopted the behaviour B right.

So, it validates that it is actually difficult for any new behaviour or action to cascade on a network. Now, in the next screen cast will see that how as we keep increasing the payoff associated with this action A, a graph ultimately the cascading in a graph is the cascading of action A in the graph is more and rather it can create a complete cascade also.